DEFINIÇÃO

Instalação do PostgreSQL. Tipos de dados. Criação de tabelas. Manipulação de linhas. Controle de transação.

PROPÓSITO

Compreender a instalação do PostgreSQL é importante para conhecer um ambiente computacional típico de banco de dados em ambientes corporativos. Entender sobre tipos de dados é a base para escolhas compatíveis com a natureza dos dados a serem armazenados. Para desenvolver sistemas com uso de banco de dados, é necessário conhecer os comandos para manipulação de linhas nas tabelas, além de identificar como eles são controlados a partir do conceito de transação.

PREPARAÇÃO

Antes de iniciar o conteúdo deste tema, certifique-se de ter baixado o SGBD PostgreSQL em seu computador.

OBJETIVOS

MÓDULO 1

Compreender o processo de instalação do PostgreSQL

MÓDULO 2

Empregar comandos para criação e alteração de tabelas

MÓDULO 3

Empregar comandos para manipular linhas nas tabelas

MÓDULO 4

Empregar comandos de controle de transação

INTRODUÇÃO

Ao longo deste tema, vamos analisar as características básicas do sistema gerenciador de banco de dados (SGBD) PostgreSQL, envolvendo as etapas utilizadas para instalar esse

SGBD no Linux e no Windows.

O PostgreSQL é um SGBD de código aberto desenvolvido em linguagem C e está disponível para ser utilizado em diversos ambientes de sistemas operacionais, tais como Linux, Unix, Windows, OS X, Solaris, entre outros.

Vamos explorar vários recursos da linguagem SQL, com foco na aprendizagem de comandos classificados como **DDL**. Aprenderemos também comandos **CRUD**, sigla em inglês que faz referência a quatro operações básicas: criação, consulta, atualização e remoção de dados, respectivamente.

Por fim, vamos entender que, internamente, o SGBD trata de diversas operações de maneira atômica, ou seja, um conjunto de comandos deve ser executado como uma unidade lógica de trabalho, ou nenhuma operação deve ser realizada. Trata-se do conceito de transação. Aprenderemos, então, os principais comandos que gerenciam transações.

Clique aqui para baixar o arquivo com todos os códigos que serão utilizados nas consultas dos módulos deste tema.

DDL

Data Definition Language, ou Linguagem de Definição de Dados.

CRUD

Create – Read – Update – Delete.

MÓDULO 1

- Compreender o processo de instalação do PostgreSQL

BREVE HISTÓRICO

O PostgreSQL surgiu a partir de um projeto denominado POSTGRES, assim denominado por ser originário do projeto INGRES (Post INGRES), de responsabilidade da Universidade da Califórnia em Berkeley.

A implementação do POSTGRES foi iniciada em 1986, tornando-se operacional em 1987. Sua primeira versão foi lançada ao público externo em 1989. Nos dois anos seguintes, foram lançadas a segunda e terceira versões.

Em 1995, foi disponibilizado o Postgres95, com revisão no código do projeto e a adoção da linguagem **SQL** como interface padrão.

Em 1996, o produto foi renomeado para PostreSQL, começando pela versão 6, considerado continuidade do Postgres95, a versão 5. O projeto ganhou visibilidade e, atualmente, o PostgreSQL é conhecido como um dos principais SGBDs de código aberto, com versões para Windows, Mac OS e Linux.

SQL

Structured Query Language, ou Linguagem de Consulta Estruturada.

Wright Studio/Shutterstock

ARQUITETURA DO POSTGRESQL

Antes de instalarmos o PostgreSQL, é importante entendermos sua arquitetura básica. O PostreSQL utiliza o modelo cliente-servidor. Sob esse contexto, destacamos os seguintes processos que cooperam entre si:

Processo servidor, responsável por funções, tais como:
Gerenciar os arquivos do banco de dados
Gerenciar as conexões entre os aplicativos e o SGBD
Avaliar e executar no banco de dados os comandos submetidos pelos clientes
Aplicativo cliente do usuário, responsável por funções, tais como:
Solicitar acesso ao SGBD

	Enviar comandos para manipulação de linhas em tabelas. A manipulação pode envolver, por exemplo, inserção, alteração ou mesmo remoção de dados
	Enviar comandos para consulta em uma ou diversas tabelas, com objetivo de recuperar informações do banco de dados

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Em um ambiente cliente-servidor, tanto o cliente quanto o servidor podem estar localizados em máquinas diferentes, em uma rede local ou mesmo geograficamente distantes.

Em geral, a comunicação entre cliente e servidor ocorre por meio de uma conexão de rede utilizando o protocolo **TCP/IP**. Esse protocolo tem por objetivo a padronização das comunicações em rede, em especial as comunicações na Web.

TCP/IP

O TCP/IP é um conjunto de protocolos de comunicação entre computadores em rede. Seu nome faz referência a dois protocolos: Transmission Control Protocol / Internet Protocol.

Fonte: Wikipedia

O PostgreSQL suporta várias conexões simultâneas de clientes. Para isso, um processo para cada conexão é iniciado. Em seguida, o cliente e o novo processo realizam comunicação.

Como instalar o PostgreSQL em seu computador?

RESPOSTA

Recomendamos que você acompanhe as versões do PostgreSQL na página oficial do produto para escolher uma versão de acordo com seu sistema operacional e que lhe interesse.

INSTALAÇÃO DO POSTGRESQL NO LINUX

Para instalação em Linux com código fonte, você pode executar os passos a seguir:

FAZER DOWNLOAD DO ARQUIVO .TAR.GZ

Se considerarmos o Ubuntu, o download pode ser realizado a partir do site do PostgreSQL, na seção relativa a esse sistema operacional. Usaremos o qualificador “--" para comentários a respeito dos comandos.

OBTER O CÓDIGO FONTE

No prompt do Linux, executar os dois comandos a seguir:

gunzip postgresql-12.3.tar.gz

-- descompacta o arquivo .gz gerando o arquivo .tar

tar xf postgresql-12.3.tar

-- abre o arquivo .tar criando o diretório postgresql-12.3

ACESSAR O DIRETÓRIO CRIADO PELO TAR

Após a conclusão da etapa anterior, deve-se executar o comando a seguir para ter acesso ao diretório criado pelo tar:

cd postgresql-12.3

-- cd (change directory)

REALIZAR O PROCESSO DE INSTALAÇÃO

Após a conclusão da etapa anterior, é preciso executar os comandos a seguir para realizar o processo de instalação do PostgreSQL:

./configure

-- script para configurar a árvore de diretórios (cria o diretório /usr/local/psql)

gmake

-- GNU make: inicializa o build, pode levar de 5 a 30 minutos e termina com a mensagem:

– All of PostgreSQL is successfully made. Ready to install.

su

-- muda login de usuário para o superusuário root (pede a senha do root)

gmake install

-- realiza a instalação como root

CHECAR A INSTALAÇÃO

Após a conclusão da etapa anterior, deve-se executar os comandos a seguir para checar instalação do PostgreSQL:

adduser postgres
-- cria usuário postgres, superusuário do PostgreSQL (seuseradd no Fedora)
mkdir /usr/local/pgsql/data
-- cria o diretório data onde ficarão as bases de dados
chown postgres /usr/local/pgsql/data
-- muda o dono do diretório data para postgres
su - postgres
-- muda login de usuário para postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
-- cria um grupo de BD no diretório data
/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
-- inicializa o servidor em segundo plano
/usr/local/pgsql/bin/createdb test
-- cria um database test
/usr/local/pgsql/bin/psql test
-- inicia uma sessão no PostgreSQL, usando a interface de linha de comandos psql, como usuário postgres no database test

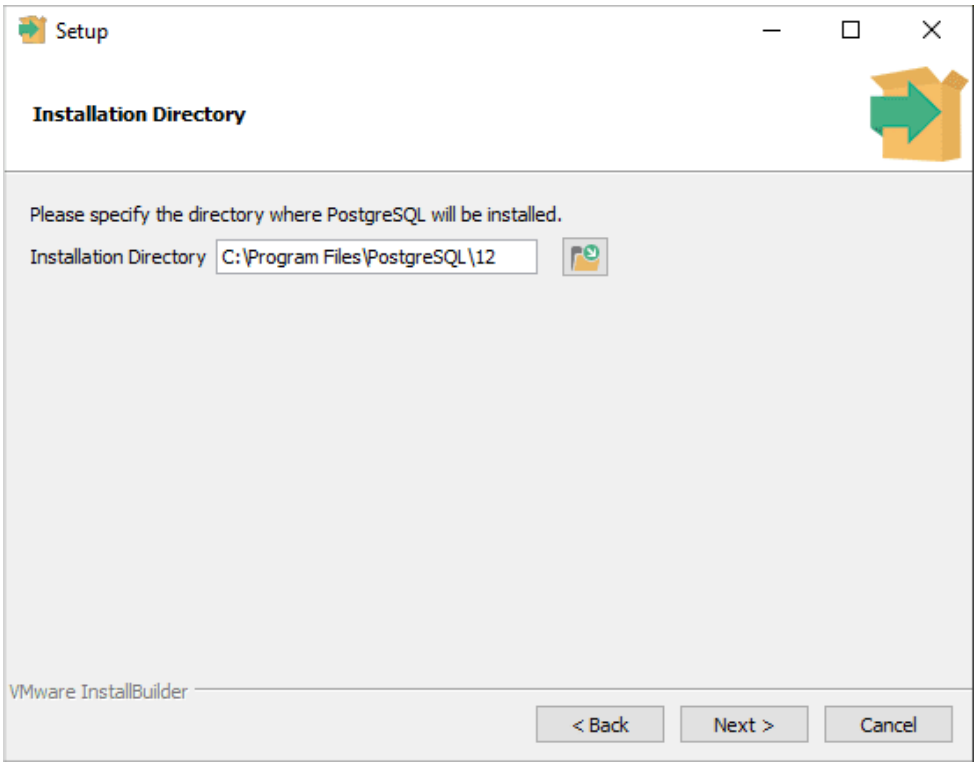
INSTALAÇÃO DO POSTGRESQL NO WINDOWS

📢 ATENÇÃO

O procedimento para instalação do PostgreSQL no sistema operacional Windows é bastante trivial e basicamente segue o padrão *(Next → Next → ... → Finish)*.

Após fazer o download do arquivo instalador para Windows (postgresql-12.3-1-windows-x64.exe, com cerca de 195 MB, no caso da versão 12), deve-se executar o arquivo como

usuário administrador. Após a tela de inicialização da instalação, será mostrada a de localização do diretório de instalação que, por padrão, criará a pasta C:\Program Files\PostgreSQL\12, onde 12 é a versão do PostgreSQL.



Fonte: Software de instalação do PostgreSQL para Windows

📷 Tela de definição do diretório de instalação.

A seguir, o instalador perguntará quais componentes serão instalados junto com o servidor PostgreSQL. Por padrão, são instalados:

O PGADMIN

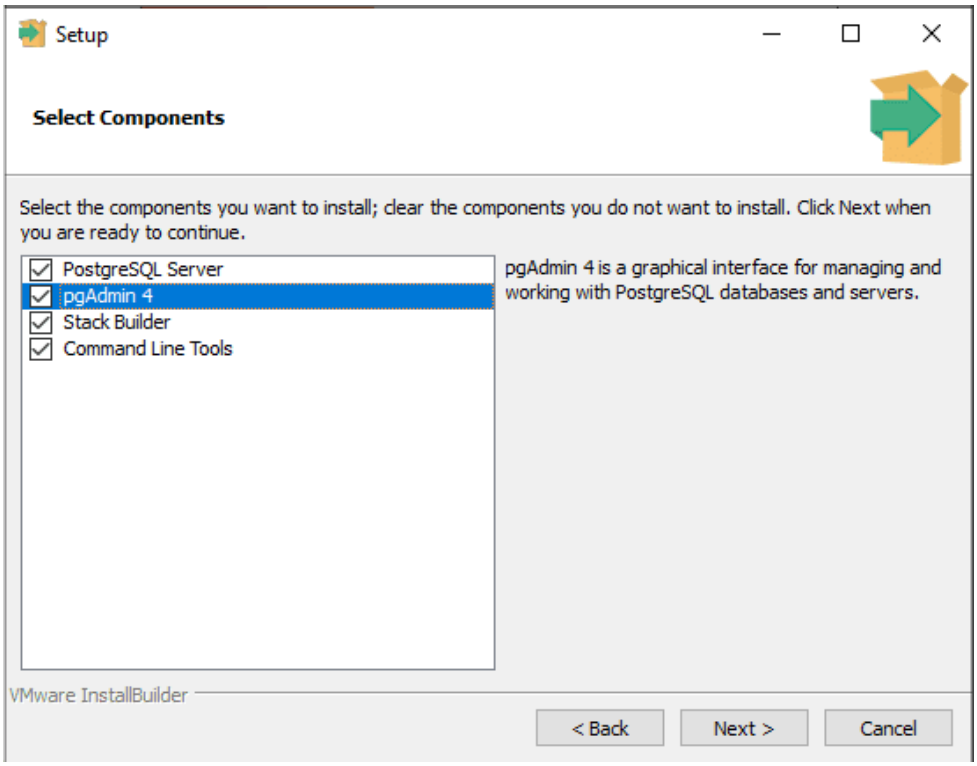
Uma interface gráfica de administração;

O PSQL

Uma ferramenta de linha de comando para administração;

O STACK BUILDER

Uma ferramenta útil para gerenciar a instalação de módulos complementares, tais como utilitários, drivers e extensões.



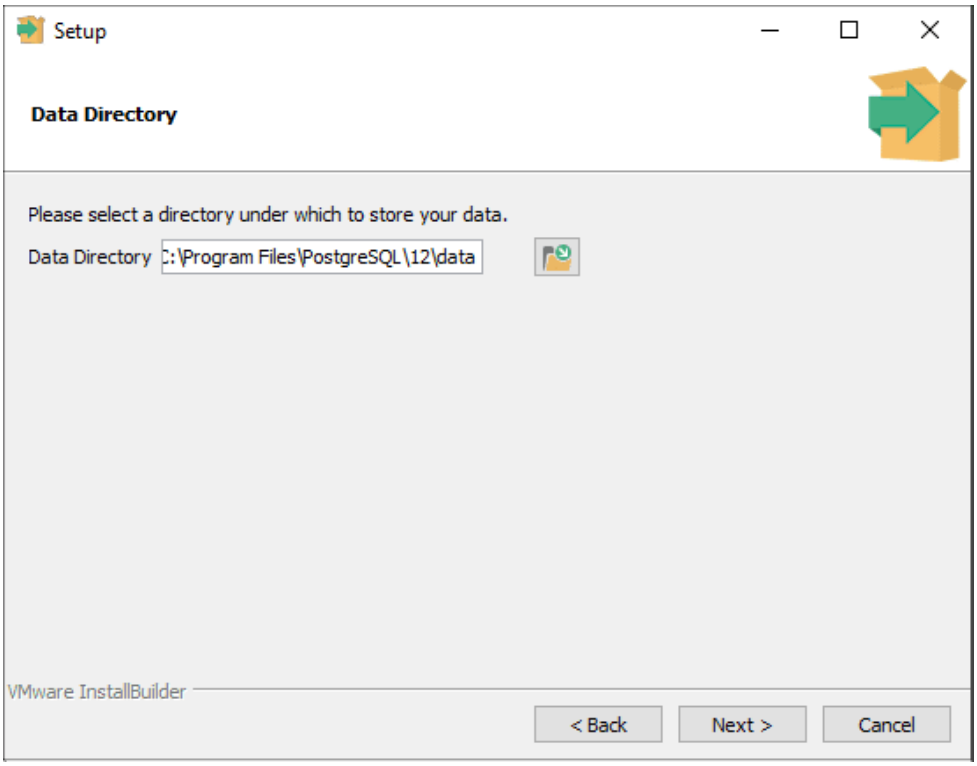
Fonte: Software de instalação do PostgreSQL para Windows

📷 Tela de seleção de componentes a instalar.

Em seguida, o instalador determina o diretório onde ficarão armazenados os dados no seu servidor. Na instalação padrão do Windows, os dados ficam armazenados no diretório C:\Program Files\PostgreSQL\12\data. Nesse diretório, é criado um subdiretório C:\Program Files\PostgreSQL\12\data\base, dentro do qual será criada uma pasta numerada para cada *database*, a começar pelo *database* padrão do servidor, denominado postgres, criado com a instalação.

📢 ATENÇÃO

Cada pasta correspondente a um *database* armazena arquivos numerados contendo metadados do catálogo do SGBD, assim como arquivos numerados para cada tabela criada dentro do *database*.



Fonte: Software de instalação do PostgreSQL para Windows

📷 Tela de localização do diretório de dados.

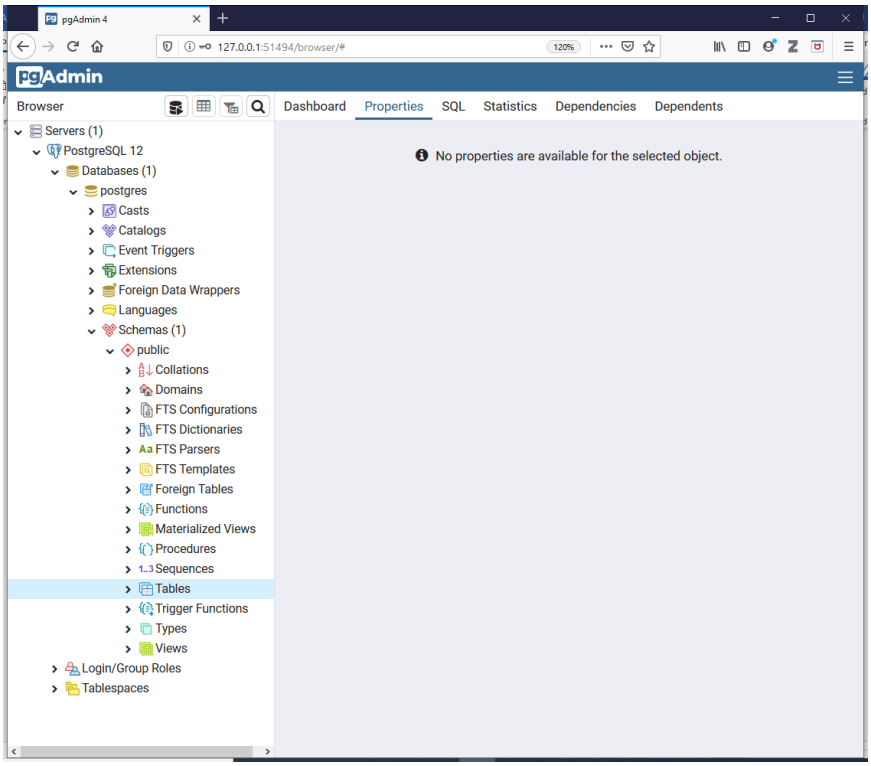
Concluído o processo de instalação do PostgreSQL, será possível visualizar no pgAdmin 4 a árvore de diretórios da instalação padrão, contendo:

Servers (1) → PostgreSQL 12

Databases (1) → postgres

Schemas (1) → public

Tables



Fonte: Software pgAdmin 4

📷 Tela do pgAdmin 4 com a árvore de diretórios da instalação padrão do PostgreSQL.

INTERFACES PARA INTERAGIR COM O POSTGRESQL

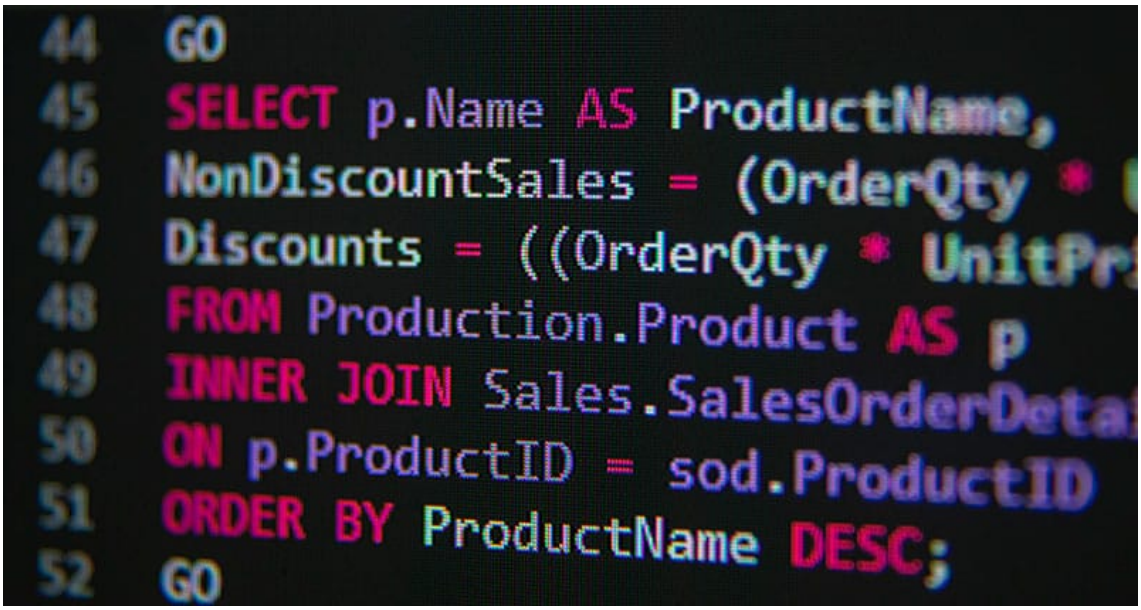
📢 ATENÇÃO

Ao longo deste e dos próximos módulos, será necessário o uso de algum tipo de interface que permita conexão ao servidor e, em seguida, o acesso aos objetos de interesse.

Além do pgAdmin 4, a interface gráfica própria que provê acesso aos recursos do SGBD via navegador, o PostgreSQL disponibiliza o psql, uma interface de linha de comando sobre a qual o usuário submete interativamente comandos ao SGBD, via terminal.

O PostgreSQL possui uma excelente documentação disponível on-line, aplicável tanto para instalação em Linux quanto para Windows, considerada uma verdadeira enciclopédia de bancos de dados relacionais. Essa documentação é válida para uso dos recursos do PostgreSQL através de quaisquer interfaces.

Alternativamente, você pode optar por baixar e usar interfaces projetadas por outros desenvolvedores. Por exemplo, o aplicativo *DBeaver* possui uma versão livre com excelentes funcionalidades. Trata-se de um aplicativo útil no desenvolvimento de atividades de administração de banco de dados.



Fonte: EvalCo/Shutterstock

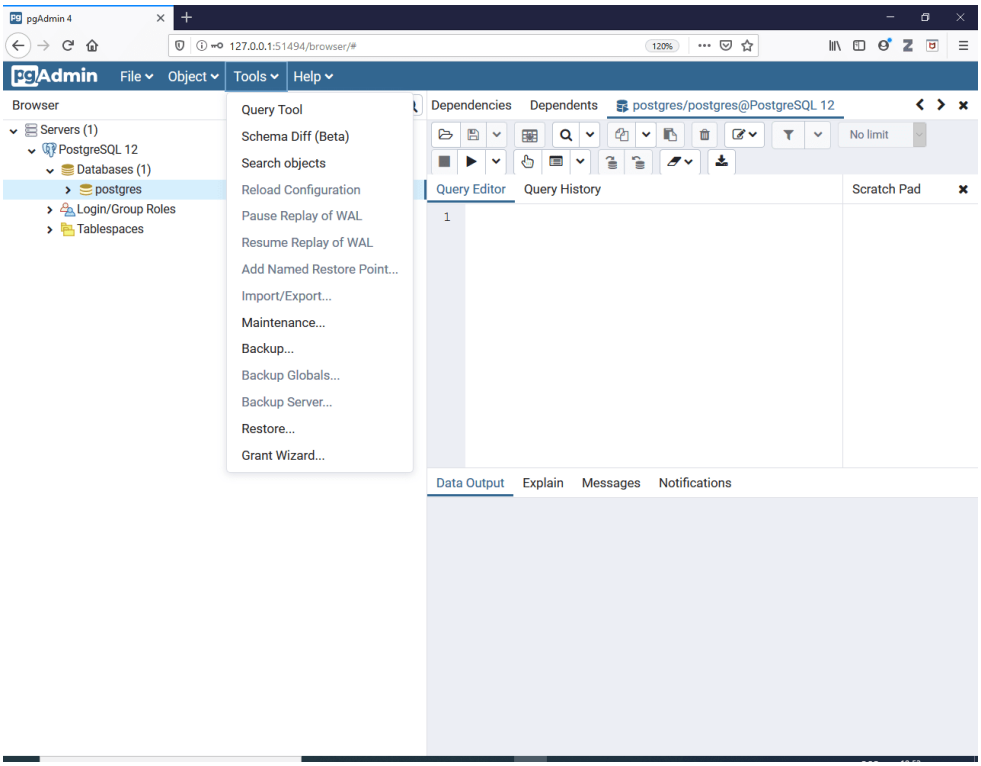
CRIANDO *DATABASES* COM O PGADMIN 4 E COM O PSQL

Tendo instalado o PostgreSQL, para nos certificarmos de que o SGBD está funcionando de maneira adequada, realizaremos um teste envolvendo a criação de *databases*, conforme a seguir:

database BDTESTEPGADMIN, a ser criado usando o pgAdmin 4.
database BDTESTEPSQL, a ser criado usando o psql

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

No Windows, selecione o botão Iniciar, digite “pgAdmin 4” e tecle <enter>. Em seguida, o navegador será aberto, e você terá acesso a um ambiente onde aparece um único *database* denominado *postgres*, criado pelo instalador.



Fonte: Software pgAdmin 4

📷 Tela do pgAdmin 4 com o *database postgres* criado pelo instalador.

Com o foco no *database postgres*, clique em Tools e em Query Tool para abrir um editor *(Query Editor)* onde você codificará e submeterá (utilizando a tecla F5 ou o botão correspondente com uma seta) comandos SQL ao servidor.

Vamos criar o *database* BDTESTEPGADMIN a partir do ambiente do pgAdmin 4. Digite o código a seguir no *Query Editor* e, em seguida, pressione a tecla F5 para executar o comando:

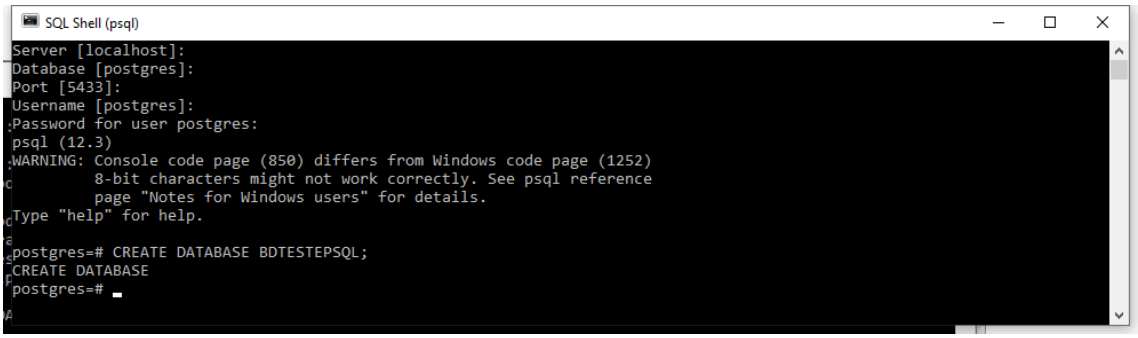
CREATE DATABASE BDTESTEPGADMIN;

Alternativamente, o *database* poderia ser criado usando a interface do pgAdmin 4, pressionado o botão direito do mouse sobre a pasta Databases e clicando em Create.

Agora, vamos criar um *database* usando a interface de terminal psql. No Windows, a partir do botão Iniciar, digite “psql” e tecle <enter>. Logo um terminal será aberto e você executará o seguinte comando (teclando <enter> após o comando):

CREATE DATABASE BDTESTEPSQL;

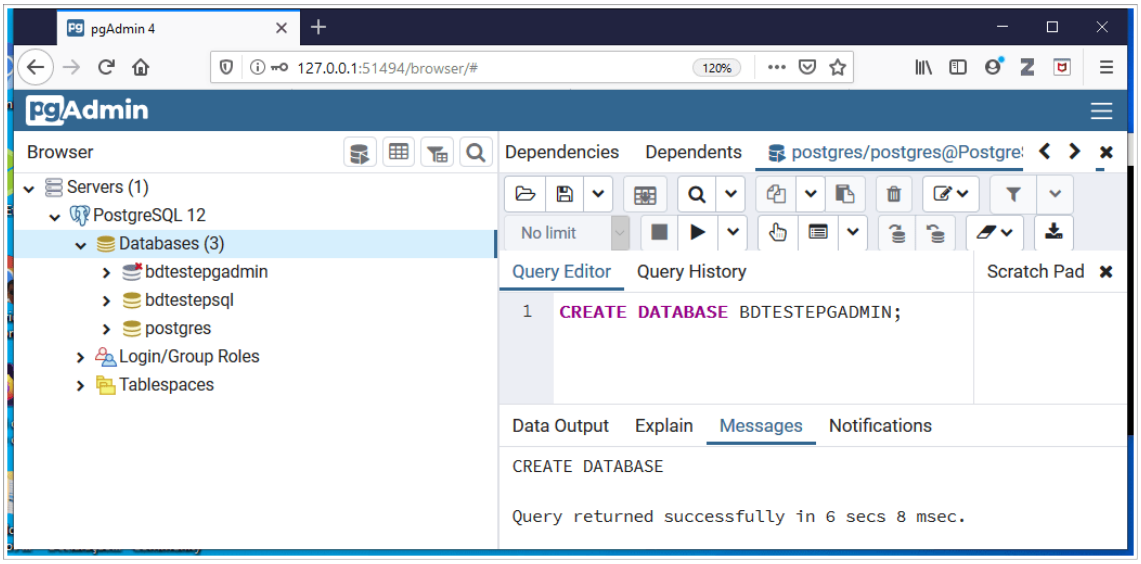
Após o SGBD executar o comando, a tela do psql ficará conforme vemos a seguir:



Fonte: Software psql

📷 Tela do psql após criação do *database* BDTESTEPSQL.

Ao final desse processo, podemos verificar, no pgAdmin 4, a criação dos dois *databases* conforme mostrado na imagem abaixo.

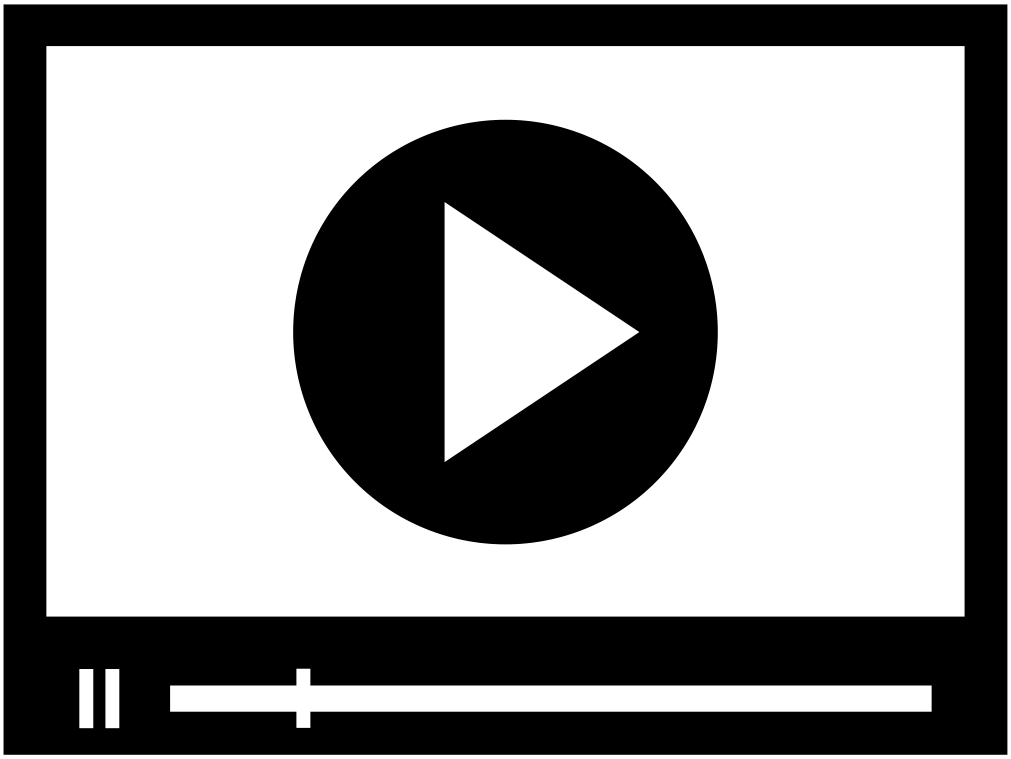


Fonte: Software psql

📷 Tela do pgAdmin 4 mostrando os *databases* recém-criados

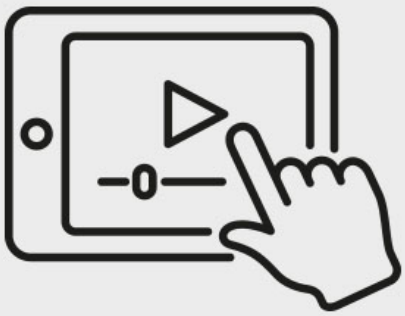
Neste módulo, apresentamos um breve histórico a respeito do PostgreSQL, além de detalhes sobre a arquitetura deste SGBD. Em seguida, mostramos o processo de instalação do SGBD no Linux e no Windows. Finalmente, verificamos a criação de *databases* utilizando as ferramentas pgAdmin 4 e psql, com objetivo de confirmarmos que a instalação foi realizada de maneira correta.

Maiores detalhes sobre a utilização do pgAdmin 4 e do psql podem ser pesquisados nas respectivas documentações que acompanham os softwares.



CRIAÇÃO DE *DATABASES* NO POSTGRESQL

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. ACERCA DO SGBD POSTGRESQL, ASSINALE A PROPOSIÇÃO VERDADEIRA.

A) O PostgreSQL é um SGBD comercial de código fechado, disponível apenas para ambiente Windows.

B) O PostgreSQL é um SGBD de código aberto, com versões compatíveis com diversos sistemas operacionais, tais como Windows, MAC OS e diversas distribuições Linux.

C) O PostgreSQL é um SGBD puramente orientado a objeto.

D) O PostgreSQL é um SGBD puramente relacional.

2. ANALISE AS SEGUINTES PROPOSIÇÕES A RESPEITO DO POSTGRESQL:

I - O COMANDO “CREATE USER BTESTE SUPERUSER INHERIT CREATEDB CREATEROLE; ” CRIA UM BANCO DE DADOS DENOMINADO BTESTE.
II - AO INSTALAR O POSTGRESQL VERSÃO 12 NO WINDOWS, A PASTA PADRÃO DE INSTALAÇÃO É “C:/PROGRAM FILES/POSTGRESQL/12”.
III - O COMANDO “CREATE TABLE CLIENTE (CODIGOC INT NOT NULL, NOME CHAR(80), CONSTRAINT CHAVECLIENTE PRIMARY KEY (CODIGOC));” CRIA UMA TABELA DENOMINADA CLIENTE COM AS COLUNAS CODIGOC E NOME, SENDO QUE A COLUNA CODIGOC É CHAVE ESTRANGEIRA.

IDENTIFIQUE AS PROPOSIÇÕES FALSAS:

A) I, II e III.

B) I e III.

C) I e II.

D) II e III.

GABARITO

1. Acerca do SGBD PostgreSQL, assinale a proposição verdadeira.

A alternativa **"B "** está correta.

De fato, o PostgreSQL é um SGBD livre e está disponível para funcionamento em diversas plataformas de sistemas operacionais.

2. Analise as seguintes proposições a respeito do PostgreSQL:

I - O comando “create user bteste superuser inherit createdb createrole; ” cria um banco de dados denominado bteste.
II - Ao Instalar o PostgreSQL versão 12 no Windows, a pasta padrão de instalação é “C:/Program Files/PostgreSQL/12”.
III - O comando “create table cliente (codigoc int not null, nome char(80), constraint chavecliente primary key (codigoc));” cria uma tabela denominada cliente com as colunas codigoc e nome, sendo que a coluna codigoc é chave estrangeira.

Identifique as proposições falsas:

A alternativa **"B "** está correta.

O comando expresso na primeira proposição cria um usuário denominado bteste, e não um banco de dados. O comando expresso na terceira proposição cria uma tabela denominada cliente. No entanto, a coluna codigoc é chave primária, e não chave estrangeira.

MÓDULO 2

⦿ Empregar comandos para criação e alteração de tabelas

BREVE HISTÓRICO DA SQL

A SQL foi criada na IBM na década de 1970, sendo originalmente chamada de SEQUEL (Strucutured English Query Language) , inspirada, principalmente, na aparente facilidade de uso do comando SELECT para consulta a tabelas dos bancos de dados relacionais.

Com a evolução dos sistemas gerenciadores de banco de dados (SGBDs), diversas empresas lançaram produtos incorporando funcionalidades à SQL, o que ocasionou problemas de compatibilidade. Buscando uma solução, o instituto **ANSI** definiu padrões para a linguagem SQL, a qual passou a ser referenciada ANSI-SQL.

Atualmente, há diversos SGBDs compatíveis com o padrão ANSI SQL, que vai muito além de consultas com o comando SELECT, englobando sublinguagens para definição de dados (CREATE, ALTER, DROP) e para manipulação de dados (INSERT, UPDATE, DELETE), além de comandos de controle típicos para administração do banco de dados. Ao mesmo tempo, vários produtos de SGBDs relacionais apresentam extensões à linguagem, como modo de facilitar o dia a dia dos desenvolvedores.

ANSI

American National Standards Institute, também conhecido por sua sigla ANSI, é uma organização particular norte-americana sem fins lucrativos que tem por objetivo facilitar a padronização dos trabalhos de seus membros.

Fonte: Wikipedia

ACESSO AO POSTGRESQL

Quero criar tabelas em um SGBD PostgreSQL. Por onde começo?

Executando o pgAdmin 4, o navegador será aberto, e você terá acesso ao ambiente que permite manipular os objetos do PostgreSQL.

Depois, escolha um *database* na hierarquia e dê um clique com o botão inverso do mouse. Em seguida, escolha a opção *Query Tool*.

DICA

O pgAdmin é a interface Web padrão do PostgreSQL, mas você pode escolher o utilitário de sua preferência para praticar os comandos que aprenderemos ao longo das próximas seções.

CRIANDO UM BANCO DE DADOS

Após escolher um utilitário para acessar o servidor PostgreSQL, será necessário criar um *database* para, em seguida, manipular tabelas. Por exemplo, para criarmos um *database* denominado bdestudo, é necessário executar o comando a seguir:

-- Comando para criar um *database*.
CREATE DATABASE BDESTUDO;

No comando acima, a linha com "--" corresponde a comentário e seu conteúdo não é processado pelo SGBD. Caso haja necessidade de remover o *database* bdestudo, basta executar o comando a seguir:

-- Comando para remover um *database*.
DROP DATABASE BDESTUDO;

Antes de prosseguirmos com a criação de tabelas, principal objetivo deste módulo, é preciso compreender que todo *database* criado no PostgreSQL possui um *schema* padrão denominado *public*, onde as tabelas a serem criadas no *database* serão armazenadas. Assim, se não especificarmos a qual *schema* do *database* pertence uma tabela que estamos criando, esta será armazenada no *schema public*. Para especificarmos um *schema* diferente do *public*, antes de criar uma tabela, devemos criar o respectivo *schema*, com o comando CREATE SCHEMA.

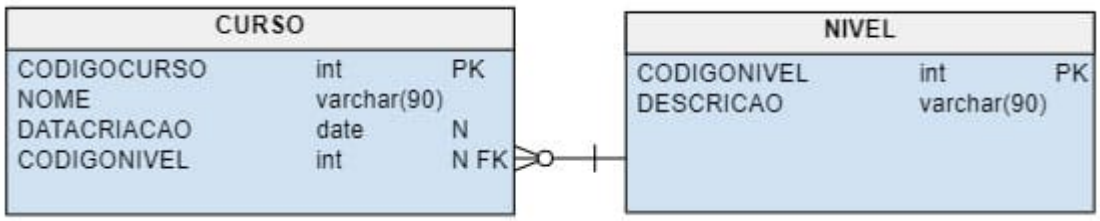
★ EXEMPLO

CREATE SCHEMA esquema;

A partir de então, qualquer tabela pertencente ao *schema* esquema deverá ser especificada pelo seu nome completo: esquema.tabela

CRIANDO TABELAS

Já sabemos que um banco de dados, em geral, possui diversas tabelas. As tabelas são criadas com auxílio do comando CREATE TABLE. Usaremos esse comando para implementar o modelo expresso na figura a seguir, dentro do *database* bdestudo anteriormente criado:



Fonte: Fonte: O autor

📷 Modelo relacional com as tabelas CURSO e NIVEL.

Veja a seguir a sintaxe **básica** do comando CREATE TABLE:

CREATE TABLE NOMETABELA (
COLUNA1 - TIPODEDADOS [RESTRIÇÃO],
COLUNAN - TIPODEDADOS [RESTRIÇÃO],
PRIMARY KEY (COLUNA),
FOREIGN KEY (COLUNA) REFERENCES NOMETABELA (COLUNA)
CONSTRAINT RESTRIÇÃO);

Vamos agora analisar o significado de cada item na sintaxe apresentada anteriormente:

NOMETABELA	representa o nome da tabela que será criada

COLUNA1 e COLUNAN	representa a(s) coluna(s) da tabela
TIPODEDADOS	indica tipo de dados ou domínio da coluna
RESTRIÇÃO	aponta alguma propriedade associada à coluna em questão. Por exemplo, podemos definir se a coluna é obrigatória ou opcional
PRIMARY KEY	indica a coluna, ou conjunto de colunas, representativa da chave primária
FOREIGN KEY	sinaliza a coluna, ou conjunto de colunas, com restrição de chave estrangeira
CONSTRAINT RESTRIÇÃO	indica alguma restrição que poderá ser declarada
A sintaxe **completa** a respeito do comando CREATE TABLE no PostgreSQL pode ser encontrada no site oficial do PostgreSQL. Ao final da sintaxe, são descritas as características compatíveis com o padrão SQL.	

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

TIPOS DE DADOS

Cada coluna de tabela deve pertencer a um tipo de dados. No PostgreSQL, os tipos mais comuns são:

bigint	valores inteiros compreendidos entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807
char(comprimento)	útil para sequências de caracteres de tamanho fixo. O parâmetro comprimento determina o valor da sequência. Esse tipo de dado preenche a coluna com espaços em branco até completar o total de caracteres definidos, caso a totalidade do tamanho do campo não esteja preenchida
date	data de calendário no formato AAAA-MM-DD
decimal	determina a precisão do valor de casas decimais
double	precisão do valor de até 15 casas decimais
int ou integer	valores inteiros compreendidos entre -2.147.483.648 e 2.147.483.647
money	valores monetários compreendidos entre – 92.233.720.368.547.758.08 e 92.233.720.368.547.758.07
numeric	precisão do valor de casas decimais
real	precisão do valor de até seis casas decimais
serial	gera valor único inteiro sequencial para um novo registro

	entre 1 e 2.147.483.647
smallint	representa valores compreendidos entre 32.768 e 32.767
time	representa horário no intervalo de tempo entre 00:00:00 e 24:00:00
varchar(comprimento)	útil para sequência de dados de caracteres com comprimento variável. Não armazena espaços em branco não utilizados para compor *string* (em branco) em seu lado direito

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

DICA

Para maiores detalhes sobre os tipos de dados, sugerimos que você acesse a documentação do PostgreSQL, disponível no site oficial do produto.

EXEMPLO ENVOLVENDO CRIAÇÃO DE TABELAS

Veja a seguir o código SQL que permite a criação das tabelas NIVEL e CURSO, respectivamente:

```
1  -- Comando para criar a tabela nivel
2  CREATE TABLE NIVEL (
3     CODIGONIVEL int  NOT NULL,
4     DESCRICAO varchar(90)  NOT NULL,
5     CONSTRAINT CHAVEPNIVEL PRIMARY KEY (CODIGONIVEL)
6  );
7  -- Comando para criar a tabela curso
8  CREATE TABLE CURSO (
9     CODIGOCURSO int  NOT NULL,
10    NOME varchar(90)  NOT NULL UNIQUE,
11    DATACRIACAO date  NULL,
12    CODIGONIVEL int  NULL,
13    CONSTRAINT CHAVEPCURSO PRIMARY KEY (CODIGOCURSO),
14    FOREIGN KEY (CODIGONIVEL)  REFERENCES NIVEL (CODIGONIVEL)
15 );
```

Fonte: O autor

A tabela NIVEL está especificada no bloco de comandos entre as linhas 2 e 6. As duas colunas da tabela são obrigatórias.

A tabela CURSO está especificada no bloco de comandos entre as linhas 8 e 15. As colunas DATACRIACAO e CODIGONIVEL são opcionais. Em especial, CODIGONIVEL significa que um curso pode ser criado e, em outro momento, ser associado à informação que caracteriza o nível dele.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

📢 ATENÇÃO

Note que, como não foi especificado um *schema* para armazenar essas tabelas, elas pertencerão ao *schema* padrão *public* do *database* bdestudo.

Finalmente, observe que, na linha 5, a coluna NOME foi declarada com o qualificador UNIQUE. Na prática, o SGBD controlará a propriedade de unicidade na referida coluna, proibindo que haja mais de uma ocorrência da mesma ao longo de todo o ciclo de vida do banco de dados.

GERENCIAMENTO DE *SCRIPTS* NA PRÁTICA

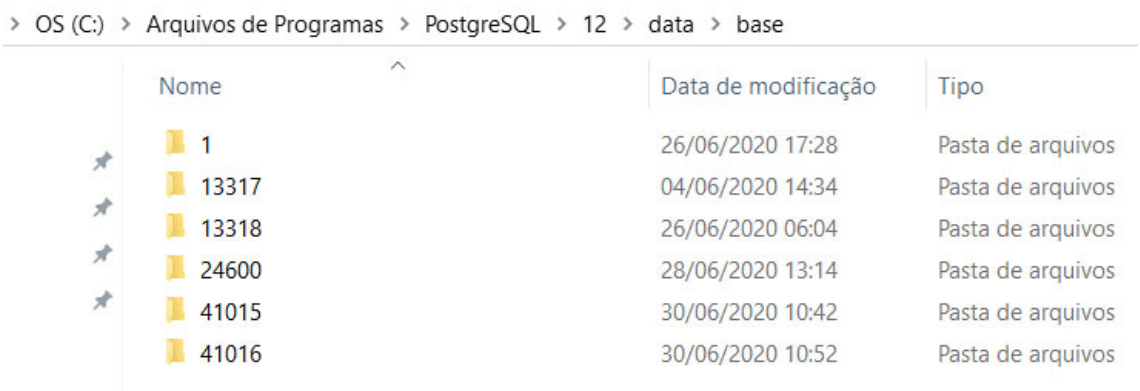
Note que o *script* SQL acima possui 15 linhas. Esse é um exemplo didático que possui somente duas tabelas. No entanto, você vai perceber que no dia a dia os projetos reais possuem quantidade de tabelas que facilmente podem ultrapassar dois dígitos.

É importante que você conheça comandos DDL, mas o uso prático ocorrerá com auxílio de ferramentas que automatizam o processo de gestão e administração de dados. Tais ferramentas permitem - manipulando elementos visuais - criar tabelas, relacionamentos e restrições, além de executar outras atividades. As ferramentas permitem gerar código DDL referente ao projeto como um todo, ou parte dele. São exemplos: DBeaver, Vertabelo, SQL Power Architect, Toad for SQL, Erwin, entre outras.

SGBD POSTGRESQL NOS BASTIDORES

Aprendemos a criar um *database* com o uso do comando CREATE DATABASE. No entanto, a submissão de um simples CREATE DATABASE ao PostgreSQL gera uma série de etapas gerenciadas pelo servidor.

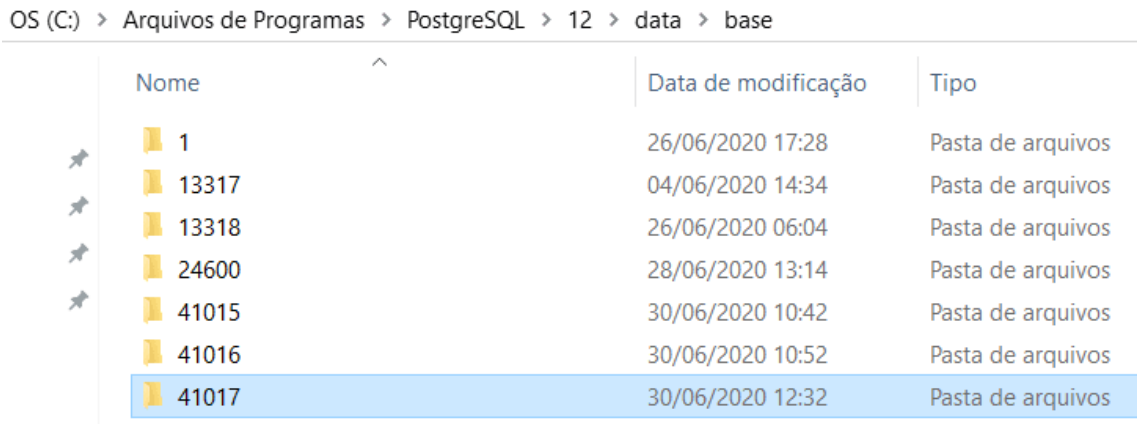
Se considerarmos a instalação padrão no Windows, ao criar um *database* o servidor cria uma pasta identificada por um número (OID), dentro do diretório C:\Program Files\PostgreSQL\12\data\base.



Fonte: O autor

📷 Pastas representativas de *databases*, antes da criação do *database* TESTEBANCO.

Após a execução do comando CREATE DATABASE TESTEBANCO; foi criada a pasta 41017, conforme imagem a seguir:



Fonte: O autor

Pastas representativas de *databases*, após da criação de TESTEBANCO.

Ainda, o PostgreSQL mantém informações sobre todos os *databases* em um *database* especial denominado catálogo, cujas tabelas possuem nomes iniciados com o prefixo PG_. Por exemplo, informações sobre os *databases* existentes em um servidor são armazenadas na tabela PG_DATABASE. Assim, caso você queira identificar o nome correspondente ao OID do PostgreSQL, basta executar o comando a seguir:

SELECT OID, DATNAME FROM PG_DATABASE;

O resultado desse comando pode ser visualizado na tabela a seguir:

	oid	datname
1	1	template1
2	13.317	template0
3	13.318	postgres
4	24.600	bdestudo
5	41.015	bdtestepgadmin
6	41.016	bdtestepsql
7	41.017	testebanco

Fonte: O autor

Resultado de consulta à tabela PG_DATABASE.

Precisamos ressaltar que, em seu computador, o resultado poderá ser diferente do apresentado, tendo em vista as operações que você tenha realizado no SGBD.

Fonte: Alexander Supertramp/Shutterstock

ALTERAÇÃO DE TABELA

Você vai se deparar com situações em que será necessário alterar a estrutura de uma tabela já existente.

Vamos estudar um exemplo?

Suponha que surgiu a necessidade de modelar a informação sobre a data de primeiro reconhecimento do curso. Podemos, então, alterar a estrutura da tabela CURSO, adicionando uma coluna opcional denominada DTRECONH. O comando ALTER TABLE é útil no contexto dessa tarefa.

A seguir, sintaxe **básica** do comando ALTER TABLE:

ALTER TABLE <NOMETABELA> ADD <COLUNA><TIPODEDADOS> ;

Na sintaxe apresentada:

<NOMETABELA>

Representa o nome da tabela sobre a qual haverá a modificação.

<COLUNA>

Representa o nome da coluna.

<TIPODEDADOS>

Representa o domínio da coluna.

A sintaxe **completa** a respeito do comando ALTER TABLE pode ser encontrada no site oficial do PostgreSQL.

A seguir, veja o código SQL que permite a alteração da tabela CURSO:

```
-- Comando para alterar a tabela CURSO, adicionando coluna DTRECONH
ALTER TABLE CURSO ADD DTRECONH DATE;
```

Por padrão, o SGBD cria a coluna como opcional. É como se o comando estivesse com o qualificador NULL declarado imediatamente antes do “;”.

E se desejarmos remover uma coluna da tabela? Podemos usar a sintaxe a seguir:

```
ALTER TABLE <NOMETABELA> DROP <COLUNA> ;
```

Vamos agora remover a coluna DTRECONH da tabela CURSO. A seguir, código SQL que permite a alteração:

```
-- Comando para alterar a tabela CURSO, removendo a coluna DTRECONH
ALTER TABLE CURSO DROP DTRECONH ;
```

REMOÇÃO DE TABELA

Você vai se deparar com situações em que será necessário remover uma tabela do banco de dados. Esta ação é realizada com o auxílio do comando DROP TABLE, cuja sintaxe está expressa a seguir:

```
DROP TABLE <NOMETABELA>;
```

Vamos agora remover a tabela CURSO com o devido código SQL:

-- Comando para remover a tabela CURSO

DROP TABLE CURSO;

Agora, vamos conhecer alguns cuidados que devem ser tomados quando formos manipular a estrutura de tabelas relacionadas.

CRIAÇÃO E ALTERAÇÃO DE TABELAS RELACIONADAS

No exemplo envolvendo criação de tabelas, o relacionamento entre as tabelas NIVEL e CURSO foi declarado no bloco CREATE TABLE da tabela CURSO (linha 14). No entanto, nós poderíamos ter optado por criar as tabelas NIVEL e CURSO sem relacionamento, para, em seguida, alterar a tabela CURSO, adicionando a restrição de chave estrangeira.

Na hipótese dessa estratégia, o *script* SQL ficaria do seguinte modo:

```
1  -- comando para criar a tabela nivel
2  CREATE TABLE NIVEL (
3     CODIGONIVEL int  NOT NULL,
4     DESCRICAO varchar(90)  NOT NULL,
5     CONSTRAINT CHAVEPNIVEL PRIMARY KEY (CODIGONIVEL));
6  -- comando para criar a tabela curso
7  CREATE TABLE CURSO (
8     CODIGOCURSO int  NOT NULL,
9     NOME varchar(90)  NOT NULL UNIQUE,
10    DATACRIACAO date  NULL,
11    CODIGONIVEL int  NULL,
12    CONSTRAINT CHAVEPCURSO PRIMARY KEY (CODIGOCURSO));
13 -- comando para alterar a tabela curso, adicionando chave estrangeira
14 ALTER TABLE CURSO ADD FOREIGN KEY (CODIGONIVEL) REFERENCES NIVEL;
```

Fonte: O autor

📢 ATENÇÃO

Note que, no *script* anterior, as tabelas foram criadas sem chave estrangeira (linhas 1 a 12). Na linha 14, o comando ALTER TABLE modifica a estrutura da tabela CURSO, implementando a restrição de chave estrangeira que representa o relacionamento entre CURSO e NIVEL.

CUIDADOS AO MANIPULAR TABELAS RELACIONADAS

Aprendemos que um banco de dados relacional é composto por diversas tabelas e que cada tabela em geral possui várias colunas. Vimos também que o relacionamento entre tabelas é implementado com o uso do mecanismo de chave estrangeira.

Quando definimos que alguma coluna é uma chave estrangeira, na prática, estamos criando uma dependência entre as tabelas envolvidas, pois toda chave estrangeira aponta para o valor de alguma chave primária.

Ao mesmo tempo, todo SGBD precisa manter a integridade dos dados durante todo o ciclo de vida do banco de dados. Com isso, não basta somente conhecermos a estrutura dos comandos para alteração de tabelas.

RESUMINDO

Queremos dizer que, em algumas situações, mesmo que o comando para alteração ou exclusão esteja correto sob o ponto de vista sintático, o SGBD sempre prioriza a integridade dos dados e pode inibir sua execução caso o resultado tenha potencial para gerar inconsistência nos dados.

Vamos estudar um exemplo?

Suponha que temos interesse em remover a tabela NIVEL. Para isso, executaremos o seguinte comando SQL:

```
-- Comando para remover a tabela NIVEL
DROP TABLE NIVEL;
```

O SGBD não removerá a tabela NIVEL e retornará uma mensagem de erro, informando que há um objeto (tabela CURSO) que depende da tabela NIVEL.

Perceba que se o SGBD removesse a tabela NIVEL, a tabela CURSO ficaria inconsistente, visto que sua chave estrangeira (CODIGONIVEL) faz referência à chave primária da tabela NIVEL. Assim, antes de remover uma tabela do banco de dados, é necessário avaliar todos os relacionamentos desta.

E se ainda assim quiséssemos remover a tabela NIVEL?

RESPOSTA

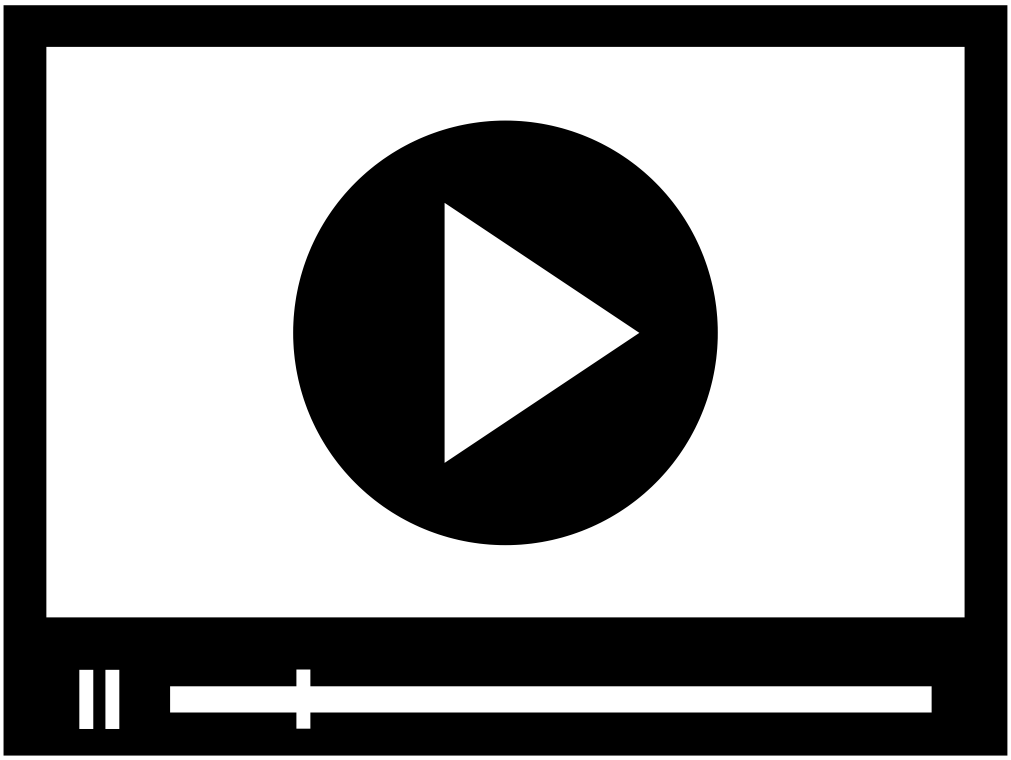
Antes, precisaríamos remover as dependências, para em seguida excluí-la do banco de dados. No entanto, como, no banco de dados, pode haver várias dependências envolvendo a tabela NIVEL – o que tornaria o processo mais demorado – o SGBD dispõe de um recurso que realiza essa tarefa automaticamente. Trata-se de remoção em cascata.

Nesse caso, poderíamos usar o comando SQL a seguir:

```
-- Comando para remover a tabela NIVEL - remoção em cascata
DROP TABLE NIVEL CASCADE;
```

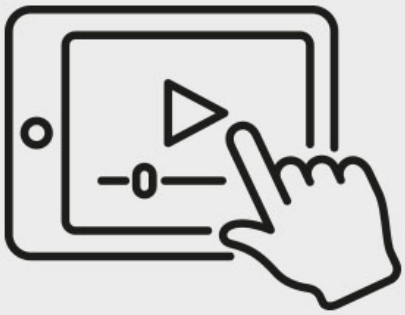
Internamente, o comando altera a estrutura da tabela CURSO, removendo a restrição de chave estrangeira da coluna CODIGONIVEL. Em seguida, a tabela NIVEL é removida do banco de dados.

Ao longo deste módulo, aprendemos comandos básicos da linguagem SQL, úteis para a criação de tabelas no SGBD PostgreSQL. Ainda, conhecemos os tipos de dados mais comuns do PostgreSQL. Em seguida, estudamos comandos para a alteração e a remoção de tabelas do banco de dados.



COMANDOS DE CRIAÇÃO, ALTERAÇÃO E REMOÇÃO DE TABELAS

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. ANALISE O *SCRIPT* A SEGUIR E ASSINALE A PROPOSIÇÃO VERDADEIRA:

```sql
CREATE TABLE CURSO (
   CODIGOCURSO int  NOT NULL,
   NOME char(90)  NOT NULL,
   DATACRIACAO date  NULL,
   CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));

CREATE TABLE DISCIPLINA (
   CODIGODISCIPLINA int  NOT NULL,
   NOME char(90)  NOT NULL,
   CARGAHORARIA int  NOT NULL,
   CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));

CREATE TABLE CURSODISCIPLINA (
   CODIGOCURSO int  NOT NULL,
   CODIGODISCIPLINA int  NOT NULL,
   CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA));

ALTER TABLE CURSODISCIPLINA ADD CONSTRAINT CURSODISCIPLINA_CURSO
   FOREIGN KEY (CODIGOCURSO)
   REFERENCES CURSO (CODIGOCURSO) ;

ALTER TABLE CURSODISCIPLINA ADD CONSTRAINT CURSODISCIPLINA_DISCIPLINA
   FOREIGN KEY (CODIGODISCIPLINA)
   REFERENCES DISCIPLINA (CODIGODISCIPLINA) ;
```

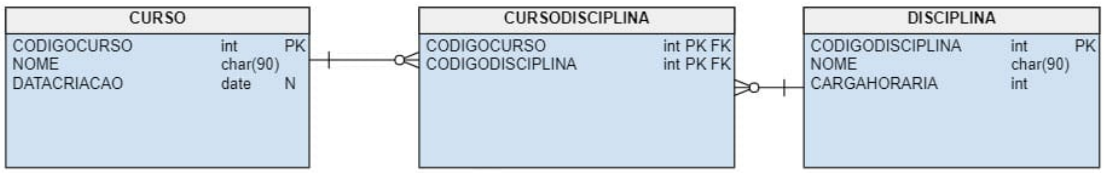
A) A execução completa do *script* cria três tabelas e todas as colunas são obrigatórias.

B) A execução dos comandos entre as linhas 1 e 16 cria três tabelas relacionadas.

C) A execução dos comandos entre as linhas 18 e 24 cria três relacionamentos.

D) A execução dos comandos entre as linhas 18 e 24 cria dois relacionamentos: o primeiro envolve as tabelas CURSODISCIPLINA e CURSO. O segundo, as tabelas CURSODISCIPLINA e DISCIPLINA.

2. ANALISE O MODELO A SEGUIR E ASSINALE A PROPOSIÇÃO VERDADEIRA:



A) A execução do *script* a seguir cria a tabela CURSODISCIPLINA e seus relacionamentos.

```
CREATE TABLE CURSODISCIPLINA (
CODIGOCURSO int NOT NULL,
CODIGODISCIPLINA int NOT NULL,
CONSTRAINT CURSODISCIPLINA_pk (PRIMARY KEY
(CODIGOCURSO,CODIGODISCIPLINA));
```

B) A execução do *script* a seguir cria a tabela DISCIPLINA.

```
CREATE TABLE CURSO (
CODIGOCURSO int NOT NULL,
NOME char(90)NOT NULL,
DATACRIACAO date NULL,
CONSTRAINT CURSO_pk (PRIMARY KEY (CODIGOCURSO));
```

C) Admitindo a existência das tabelas CURSODISCIPLINA e DISCIPLINA, a execução do *script* a seguir relaciona CURSODISCIPLINA à DISCIPLINA.

```
ALTER TABLE CURSODISCIPLINA ADD FOREIGN KEY (CODIGOCURSO)
REFERENCES CURSO (CODIGOCURSO) ;
```

D) Admitindo a existência das tabelas CURSODISCIPLINA e CURSO, a execução do *script* a seguir relaciona CURSODISCIPLINA à CURSO.

ALTER TABLE CURSODISCIPLINA ADD FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO)

GABARITO

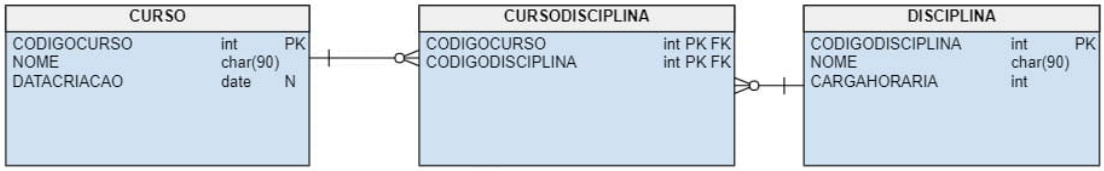
1. Analise o *script* a seguir e assinale a proposição verdadeira:

```
1  CREATE TABLE CURSO (
2     CODIGOCURSO int  NOT NULL,
3     NOME char(90)  NOT NULL,
4     DATACRIACAO date  NULL,
5     CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));
6
7  CREATE TABLE DISCIPLINA (
8     CODIGODISCIPLINA int  NOT NULL,
9     NOME char(90)  NOT NULL,
10    CARGAHORARIA int  NOT NULL,
11    CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));
12
13 CREATE TABLE CURSODISCIPLINA (
14    CODIGOCURSO int  NOT NULL,
15    CODIGODISCIPLINA int  NOT NULL,
16    CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA));
17
18 ALTER TABLE CURSODISCIPLINA ADD CONSTRAINT CURSODISCIPLINA_CURSO
19    FOREIGN KEY (CODIGOCURSO)
20    REFERENCES CURSO (CODIGOCURSO) ;
21
22 ALTER TABLE CURSODISCIPLINA ADD CONSTRAINT CURSODISCIPLINA_DISCIPLINA
23    FOREIGN KEY (CODIGODISCIPLINA)
24    REFERENCES DISCIPLINA (CODIGODISCIPLINA) ;
```

A alternativa **"D "** está correta.

De fato, há dois blocos de comando entre as linhas 18 e 24. O primeiro altera a estrutura da tabela CURSODISCIPLINA adicionando à coluna CODIGOCURSO uma restrição de chave estrangeira que implementa o relacionamento entre as tabelas CURSODISCIPLINA e CURSO. O segundo altera a estrutura da tabela CURSODISCIPLINA, adicionando à coluna CODIGODISCIPLINA uma restrição de chave estrangeira que implementa o relacionamento entre as tabelas CURSODISCIPLINA e DISCIPLINA.

2. Analise o modelo a seguir e assinale a proposição verdadeira:



A alternativa **"D "** está correta.

De fato, se observarmos o modelo, há um relacionamento entre as tabelas CURSO e CURSODISCIPLINA. Ao executar o ALTER TABLE, a coluna CODIGOCURSO da tabela CURSODISCIPLINA passa a funcionar como chave estrangeira, fazendo referência à tabela CURSO.

MÓDULO 3

⦿ Empregar comandos para manipular linhas nas tabelas

MANIPULAÇÃO DE LINHAS NAS TABELAS

Quando usamos o termo manipulação, fazemos referência às operações de inserção, atualização ou mesmo eliminação de dados. Em uma linguagem mais comercial, existe o termo **CRUD**, que representa quatro operações básicas: criação, consulta, atualização e remoção de dados, respectivamente.

No contexto da da SQL, costumamos usar o seguinte mapeamento dos comandos:

Create: *INSERT*

Read: *SELECT*

Update: *UPDATE*

Delete: *DELETE*

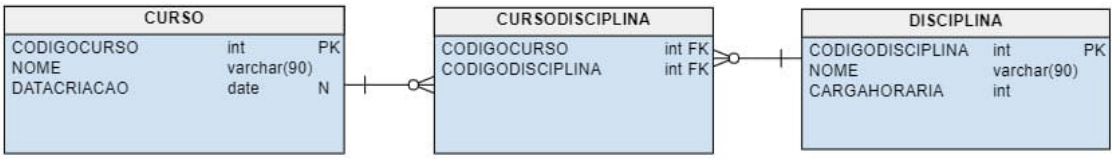
Ainda, os comandos da linguagem SQL que aprenderemos fazem parte da **DML** e são usados para inserir, modificar e remover dados.

DML

Data Manipulation Language ou Linguagem de Manipulação de Dados.

MODELO PARA OS EXEMPLOS

Ao longo da nossa aprendizagem, nós vamos admitir que o modelo a seguir está criado no banco de dados:



Fonte: O autor

Tabelas CURSO, CURSODISCIPLINA e DISCIPLINA.

DICA

Recomendamos que você crie as tabelas e insira algumas linhas, o que pode ser feito usando o script a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e ao database bdestudo criado anteriormente.

```sql
CREATE TABLE CURSO (
   CODIGOCURSO int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   DATACRIACAO date  NULL,
   CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));
CREATE TABLE DISCIPLINA (
   CODIGODISCIPLINA int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   CARGAHORARIA int  NOT NULL,
   CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));
CREATE TABLE CURSODISCIPLINA (
   CODIGOCURSO int  NOT NULL,
   CODIGODISCIPLINA int  NOT NULL,
   CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA),
   CONSTRAINT CURSODISCIPLINA_CURSO FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO) ON DELETE CASCADE ,
   CONSTRAINT CURSODISCIPLINA_DISCIPLINA  FOREIGN KEY (CODIGODISCIPLINA)  REFERENCES DISCIPLINA (CODIGODISCIPLINA) );
```

Fonte: Autor

O modelo é útil para gerenciar os dados de cursos, disciplinas e do relacionamento entre esses objetos. Em especial, cada linha da tabela CURSODISCIPLINA representa uma associação entre curso e disciplina.

INSERÇÃO DE LINHAS EM TABELA

A inserção de linhas em tabela é realizada com o auxílio do comando INSERT da SQL. Sua sintaxe **básica** está expressa a seguir:

INSERT INTO <NOMETABELA> (COLUNA1, COLUNA2,…,COLUNAn) VALUES (VALOR1, VALOR2,…,VALORn);

Na sintaxe, <NOMETABELA> representa a tabela alvo da inserção. Em seguida, são declaradas as colunas que receberão os dados; por último, os dados em si. Note que deve haver uma correspondência entre cada par COLUNA/VALOR, ou seja, o conteúdo de cada coluna deve ser compatível com o tipo de dados ou domínio dela.

A sintaxe **completa** a respeito do comando INSERT pode ser encontrada no site oficial do PostgreSQL.

Vamos estudar um exemplo?

Iremos cadastrar quatro cursos. O comando SQL a seguir pode ser utilizado:

INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO)
VALUES(1,'Sistemas de Informação','19/06/1999');
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO)
VALUES(2,'Medicina','10/05/1990');
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO)
VALUES(3,'Nutrição','19/02/2012');

```
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO)
VALUES( 4,'Pedagogia','19/06/1999');
```

📢 ATENÇÃO

Note que, dentro dos parênteses representativos dos conteúdos, o primeiro valor, por ser do tipo inteiro, foi informado sem aspas. Já o segundo valor, por ser do tipo char, foi informado com aspas. Finalmente, o valor referente ao tipo date também foi informado entre aspas. Internamente, o PostgreSQL converte o texto para o formato de data.

Agora, faremos um procedimento semelhante, cadastrando quatro disciplinas. O comando SQL a seguir pode ser utilizado:

```
INSERT INTO DISCIPLINA (CODIGODISCIPLINA,NOME,CARGAHORARIA)
VALUES( 1,''Leitura e Produção de Textos',60);
INSERT INTO DISCIPLINA (CODIGODISCIPLINA,NOME,CARGAHORARIA)
VALUES( 2,''Redes de Computadores',60);
INSERT INTO DISCIPLINA (CODIGODISCIPLINA,NOME,CARGAHORARIA)
VALUES( 3,'Computação Gráfica',40);
INSERT INTO DISCIPLINA (CODIGODISCIPLINA,NOME,CARGAHORARIA)
VALUES( 4,'Anatomia',60);
```

Agora, vamos registrar na tabela CURSODISCIPLINA algumas associações entre cursos e disciplinas. O comando SQL a seguir pode ser utilizado:

```
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,1);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,2);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,3);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (2,1);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (2,3);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (3,1);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (3,3);
```

E se submetermos ao SGBD o comando a seguir?

```
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (3,30);
```

RESPOSTA

O SGBD não realizará a inserção e retornará uma mensagem de erro informando que 30 não é um valor previamente existente na chave primária da tabela DISCIPLINA. Isso acontece porque, quando definimos (linha 16 do *script* da seção anterior) a chave estrangeira da tabela CURSODISCIPLINA, nós delegamos ao SGBD a tarefa de realizar esse tipo de validação com objetivo de sempre manter a integridade dos dados do banco de dados. Note que não existe disciplina identificada de código 30 na tabela DISCIPLINA.

MECANISMO DE CHAVE PRIMÁRIA EM AÇÃO

Já vimos que o SGBD é responsável por manter a integridade dos dados ao longo de todo o ciclo de vida do banco de dados. A consequência disso pode ser percebida ao tentarmos executar (novamente) o comando a seguir:

INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (4,'Anatomia',60);

Como já existe um registro com valor de CODIGODISCIPLINA igual a 4, o SGBD exibirá mensagem de erro informando que o referido valor já existe no banco de dados. Semelhantemente, devemos lembrar que todo valor de chave primária é obrigatório.

Vamos agora tentar inserir uma disciplina sem valor para CODIGODISCIPLINA, conforme comando SQL a seguir:

INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (NULL,'Biologia Celular e Molecular',60);

O SGBD exibirá mensagem de erro informando que o valor da coluna CODIGODISCIPLINA não pode ser nulo.

ATUALIZAÇÃO DE LINHAS EM TABELA

A atualização de linhas em tabela é realizada com o auxílio do comando UPDATE da SQL. Sua sintaxe **básica** está expressa a seguir:

```
UPDATE <NOMETABELA>
SET COLUNA1=VALOR1, COLUNA2=VALOR2,…,COLUNAn=VALORn
WHERE <CONDIÇÃO>;
```

Na sintaxe, <NOMETABELA> representa a tabela alvo da atualização. Em seguida, é declarada uma lista contendo a coluna e o seu respectivo valor novo. Por último, uma condição lógica, caso seja necessário. Isso ocorre porque, em geral, estamos interessados em alterar somente um subconjunto de linhas que é obtido a partir do processamento da cláusula WHERE. A sintaxe **completa** a respeito do comando UPDATE pode ser encontrada no site oficial do PostgresSQL.

Vamos estudar alguns exemplos?

Alteraremos para 70 a carga horária da disciplina Redes de Computadores. Para isso, basta executar o comando a seguir:

```
UPDATE DISCIPLINA SET CARGAHORARIA=70 WHERE CODIGODISCIPLINA=2;
```

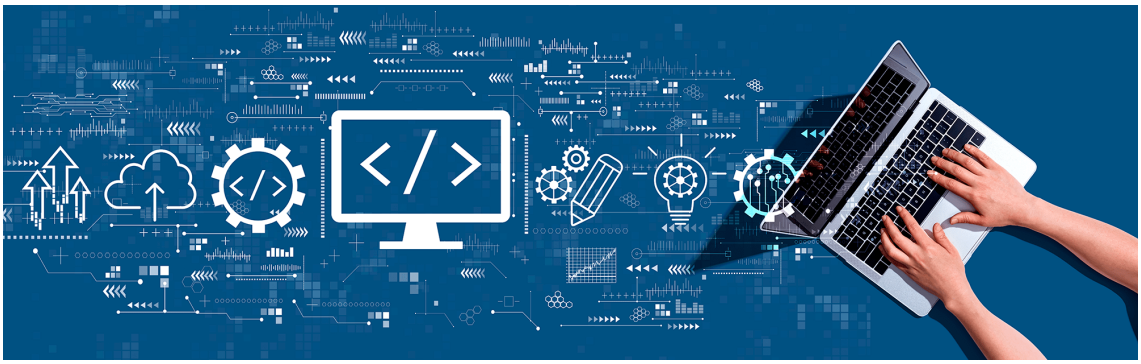
No comando, o SGBD busca na tabela a disciplina cujo valor da coluna CODIGODIISCIPLINA seja igual a 2. Em seguida, atualiza o valor da coluna CARGAHORARIA para 70. Note também que poderíamos ter executado o comando a seguir:

```
UPDATE DISCIPLINA SET CARGAHORARIA=70 WHERE NOME='Redes de Computadores';
```

Suponha agora que houve a necessidade de alterar em 20% a carga horária de todas as disciplinas cadastradas no banco de dados. Podemos executar o seguinte comando:

```
UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.2;
```

Note que, no último comando, não foi necessária a cláusula WHERE, visto que o nosso interesse era o de atualizar todas as linhas da tabela DISCIPLINA. Ainda, para obter o **novo** valor, nós utilizamos a expressão CARGAGORARIA*1.2.



Fonte: TierneyMJ/Shutterstock

ATUALIZAÇÃO DE COLUNA CHAVE PRIMÁRIA

Devemos ter especial cuidado ao planejarmos alterar o valor de coluna com o papel de chave primária em uma tabela.

Vamos supor que seja necessário alterar para 6 o valor de CODIGOCURSO referente ao curso de Pedagogia. Podemos, então, executar o comando a seguir:

UPDATE CURSO SET CODIGOCURSO=6 WHERE CODIGOCURSO=4;

Perceba que o SGBD processará a alteração, visto que não há vínculo na tabela CURSODISCIPLINA envolvendo este curso. No entanto, o que aconteceria se tentássemos alterar para 10 o valor de CODIGOCURSO referente ao curso de Sistemas de Informação?

Seguindo a mesma linha de raciocínio da última alteração, vamos submeter o comando a seguir:

UPDATE CURSO SET CODIGOCURSO=10 WHERE CODIGOCURSO=1;

📢 ATENÇÃO

O SGBD não realizará a alteração e retornará uma mensagem de erro indicando que o valor 1 está registrado na tabela CURSODISCIPLINA, coluna CODIGOCURSO. Isso significa que, se o SGBD aceitasse a alteração, a tabela CURSODISCIPLINA ficaria com dados inconsistentes, o que não deve ser permitido.

Assim, de modo semelhante ao que aprendemos na seção Mecanismo de chave primária em ação, deixaremos o SGBD realizar as alterações necessárias para manter a integridade dos dados. Vamos, então, submeter o comando a seguir:

```
ALTER TABLE CURSODISCIPLINA
DROP CONSTRAINTCURSODISCIPLINA_CURSO,
ADD CONSTRAINT CURSODISCIPLINA_CURSO
FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO)
ON UPDATE CASCADE ;
```

O que fizemos? Usamos o comando ALTER TABLE para alterar a estrutura da tabela CURSODISCIPLINA:, removemos a chave estrangeira (comando DROP CONSTRAINT) e, por último, recriamos a chave (ADD CONSTRAINT), especificando a operação de atualização (UPDATE) em cascata.

Assim, após o processamento da alteração anterior, podemos então submeter o comando, conforme a seguir:

```
UPDATE CURSO SET CODIGOCURSO=10 WHERE CODIGOCURSO=1;
```

REMOÇÃO DE LINHAS EM TABELA

A remoção de linhas em tabela é realizada com o auxílio do comando DELETE da SQL. Sua sintaxe **básica** está expressa a seguir:

```
DELETE FROM <NOMETABELA>
WHERE <CONDIÇÃO>;
```

Na sintaxe, <NOMETABELA> representa a tabela alvo da operação de deleção de linha(s). Por último, há uma condição lógica, caso seja necessário. Isso ocorre porque, em geral, estamos interessados em remover somente um subconjunto de linhas que é obtido a partir do processamento da cláusula WHERE. A sintaxe **completa** a respeito do comando DELETE pode ser encontrada no site oficial do PostgresSQL.

Vamos estudar alguns exemplos?

Suponha que temos interesse em apagar do banco de dados a disciplina de Anatomia. Podemos, então, submeter o código a seguir:

```
DELETE FROM DISCIPLINA WHERE CODIGODISCIPLINA=4;
```

O SGBD localiza na tabela DISCIPLINA a linha cujo conteúdo da coluna CODIGODISCIPLINA seja igual a 1. Em seguida, remove do banco de dados a linha em questão.

Agora vamos imaginar que tenha surgido a necessidade de remover do banco de dados a disciplina de Leitura e Produção de Textos. Podemos, então, submeter o código a seguir:

```
DELETE FROM DISCIPLINA WHERE CODIGODISCIPLINA=1;
```

O SGBD não realizará a remoção e retornará uma mensagem de erro indicando que o valor 1 está registrado na tabela CURSODISCIPLINA, coluna CODIGODISCIPLINA. Se o SGBD aceitasse a remoção, a tabela CURSODISCIPLINA ficaria com dados inconsistentes, o que não deve ser permitido.

Assim, de modo semelhante ao que aprendemos na seção Mecanismo de chave primária em ação, deixaremos o SGBD realizar as alterações necessárias para manter a integridade dos dados. Vamos, então, submeter o comando a seguir:

```
ALTER TABLE CURSODISCIPLINA
DROP CONSTRAINT CURSODISCIPLINA_DISCIPLINA,
ADD CONSTRAINT CURSODISCIPLINA_DISCIPLINA
FOREIGN KEY (CODIGODISCIPLINA) REFERENCES DISCIPLINA (CODIGODISCIPLINA)
ON DELETE CASCADE ;
```

O que fizemos? Usamos o comando ALTER TABLE para alterar a estrutura da tabela CURSODISCIPLINA:, removemos a chave estrangeira (comando DROP CONSTRAINT) e, por último, recriamos a chave (ADD CONSTRAINT), especificando operação de remoção (DELETE) em cascata.

Assim, após o processamento da alteração anterior, podemos submeter o comando conforme a seguir:

```
DELETE FROM DISCIPLINA WHERE CODIGODISCIPLINA=1;
```

Ao processar o comando, o SGBD verifica se existe alguma linha da tabela CURSODISCIPLINA com valor 1 para a coluna CODIGODISCIPLINA. Caso encontre, cada ocorrência é então removida do banco de dados.

E se quiséssemos eliminar todos os registros de todas as tabelas do banco de dados?

RESPOSTA

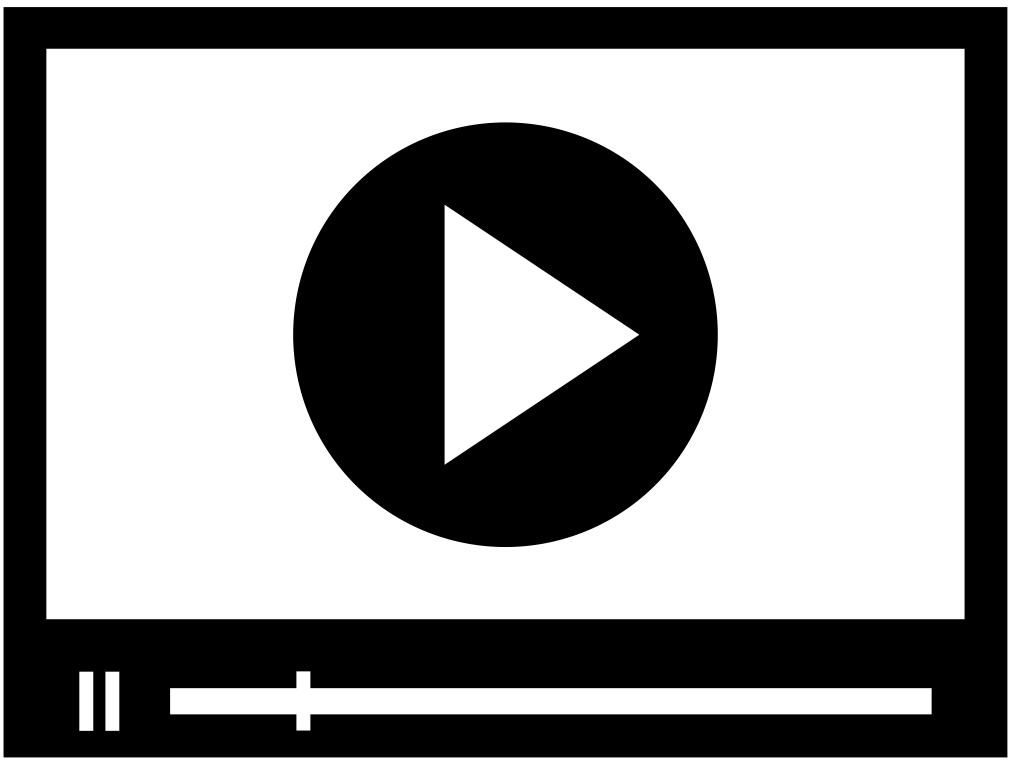
Para realizarmos esta operação, precisaremos identificar quais tabelas são mais independentes e quais são as que possuem vínculos de chave estrangeira.

No caso do nosso exemplo, CURSODISCIPLINA possui duas chaves estrangeiras, portanto, é a tabela mais dependente. As demais, não possuem chave estrangeira. De posse dessa informação, podemos submeter os comandos a seguir para completar a tarefa:

```
DELETE FROM CURSODISCIPLINA;
DELETE FROM CURSO;
DELETE FROM DISCIPLINA;
```

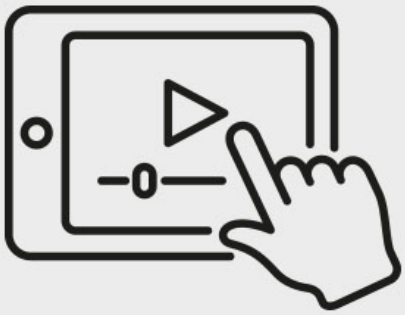
Perceba que a primeira tabela que foi usada no processo de remoção de linhas foi a CURSODISCIPLINA, pois essa é a responsável por manter as informações sobre o relacionamento entre as tabelas CURSO e DISCIPLINA. Após eliminar os registros de CURSODISCIPLINA, o SGBD removerá com sucesso os registros das tabelas CURSO e DISCIPLINA.

Ao longo deste módulo, aprendemos os comandos básicos da linguagem SQL, os quais são úteis para a manipulação de linhas no SGBD PostgreSQL. Também estudamos comandos para inserir, alterar e eliminar linhas em tabelas.



COMANDOS DE MANIPULAÇÃO DE DADOS

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. ANALISE O *SCRIPT* A SEGUIR E ASSINALE A PROPOSIÇÃO CORRETA:

```
1  CREATE TABLE CURSO (
2     CODIGOCURSO int  NOT NULL,
3     NOME char(90)  NOT NULL,
4     DATACRIACAO date  NULL,
5     CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));
6  CREATE TABLE DISCIPLINA (
7     CODIGODISCIPLINA int  NOT NULL,
8     NOME char(90)  NOT NULL,
9     CARGAHORARIA int  NOT NULL,
10    CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));
11 CREATE TABLE CURSODISCIPLINA (
12    CODIGOCURSO int  NOT NULL,
13    CODIGODISCIPLINA int  NOT NULL,
14    CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA),
15    CONSTRAINT CURSODISCIPLINA_CURSO FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO)
16    CONSTRAINT CURSODISCIPLINA_DISCIPLINA  FOREIGN KEY (CODIGODISCIPLINA)  REFERENCES DISCIPLINA (CODIGODISCIPLINA) );
17
18 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (1,'Sistemas de Informação',NULL);
19 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (2,'Medicina','10/05/1990');
20
21 INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (1,'Leitura e Produção de Textos',60);
22 INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (2,'Redes de Computadores',60);
23
24 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,1);
25 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,8);
26 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,3);
27 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (3,1);
```

A) Após a execução com sucesso do trecho entre as linhas 1 e 16, se executarmos o comando expresso na linha 18, o SGBD retornará uma mensagem de erro.

B) Após a execução com sucesso do trecho entre as linhas 1 e 16, se executarmos o comando expresso na linha 25, o SGBD retornará uma mensagem de erro.

C) Após a execução com sucesso do trecho entre as linhas 1 e 16, se executarmos o comando expresso na linha 26, o SGBD não retornará uma mensagem de erro.

D) Após a execução com sucesso do trecho entre as linhas 1 e 16, se executarmos o comando expresso na linha 25, o SGBD não retornará uma mensagem de erro.

2. ANALISE O *SCRIPT* A SEGUIR E ASSINALE A PROPOSIÇÃO CORRETA:

```
1  CREATE TABLE CURSO (
2     CODIGOCURSO int  NOT NULL,
3     NOME char(90)  NOT NULL,
4     DATACRIACAO date  NULL,
5     CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));
6  CREATE TABLE DISCIPLINA (
7     CODIGODISCIPLINA int  NOT NULL,
8     NOME char(90)  NOT NULL,
9     CARGAHORARIA int  NOT NULL,
10    CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));
11 CREATE TABLE CURSODISCIPLINA (
12    CODIGOCURSO int  NOT NULL,
13    CODIGODISCIPLINA int  NOT NULL,
14    CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA),
15    CONSTRAINT CURSODISCIPLINA_CURSO FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO) ON DELETE CASCADE ,
16    CONSTRAINT CURSODISCIPLINA_DISCIPLINA  FOREIGN KEY (CODIGODISCIPLINA)  REFERENCES DISCIPLINA (CODIGODISCIPLINA) );
17
18 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (1,'Sistemas de Informação',NULL);
19 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (2,'Medicina','10/05/1990');
20
21 INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (1,'Leitura e Produção de Textos',60);
22 INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (2,'Redes de Computadores',60);
23
24 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,1);
25 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,2);
26 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (2,1);
27
28 DELETE FROM CURSO WHERE CODIGOCURSO=1;
```

APÓS A EXECUÇÃO, COM SUCESSO, DO TRECHO ENTRE AS LINHAS 1 E 26, SE EXECUTARMOS O COMANDO EXPRESSO NA LINHA 28, QUANTAS LINHAS SERÃO REMOVIDAS DO SGBD?

A) Nenhuma.

B) Uma.

C) Duas.

D) Três.

GABARITO

1. Analise o *script* a seguir e assinale a proposição correta:

```sql
CREATE TABLE CURSO (
   CODIGOCURSO int  NOT NULL,
   NOME char(90)  NOT NULL,
   DATACRIACAO date  NULL,
   CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));
CREATE TABLE DISCIPLINA (
   CODIGODISCIPLINA int  NOT NULL,
   NOME char(90)  NOT NULL,
   CARGAHORARIA int  NOT NULL,
   CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));
CREATE TABLE CURSODISCIPLINA (
   CODIGOCURSO int  NOT NULL,
   CODIGODISCIPLINA int  NOT NULL,
   CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA),
   CONSTRAINT CURSODISCIPLINA_CURSO FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO)
   CONSTRAINT CURSODISCIPLINA_DISCIPLINA  FOREIGN KEY (CODIGODISCIPLINA)  REFERENCES DISCIPLINA (CODIGODISCIPLINA) );

INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (1,'Sistemas de Informação',NULL);
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (2,'Medicina','10/05/1990');

INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (1,'Leitura e Produção de Textos',60);
INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (2,'Redes de Computadores',60);

INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,1);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,8);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,3);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (3,1);
```

A alternativa **"B "** está correta.

De fato, se observarmos as inserções em CURSO e DISCIPLINA, vamos perceber que os valores de chave primária são os inteiros 1 e 2 em ambos os casos. Ainda, a tabela CURSODISCIPLINA possui duas chaves estrangeiras: uma referência à tabela CURSO; a outra, referência à tabela DISCIPLINA. Portanto, o valor para CÓDIGODISCIPLINA da tabela CURSODISCIPLINA não pode ser diferente de 1 ou 2.

2. Analise o *script* a seguir e assinale a proposição correta:

```sql
1  CREATE TABLE CURSO (
2     CODIGOCURSO int  NOT NULL,
3     NOME char(90)  NOT NULL,
4     DATACRIACAO date  NULL,
5     CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));
6  CREATE TABLE DISCIPLINA (
7     CODIGODISCIPLINA int  NOT NULL,
8     NOME char(90)  NOT NULL,
9     CARGAHORARIA int  NOT NULL,
10    CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));
11 CREATE TABLE CURSODISCIPLINA (
12    CODIGOCURSO int  NOT NULL,
13    CODIGODISCIPLINA int  NOT NULL,
14    CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA),
15    CONSTRAINT CURSODISCIPLINA_CURSO FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO) ON DELETE CASCADE ,
16    CONSTRAINT CURSODISCIPLINA_DISCIPLINA  FOREIGN KEY (CODIGODISCIPLINA)  REFERENCES DISCIPLINA (CODIGODISCIPLINA) );
17
18 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (1,'Sistemas de Informação',NULL);
19 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (2,'Medicina','10/05/1990');
20
21 INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (1,'Leitura e Produção de Textos',60);
22 INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (2,'Redes de Computadores',60);
23
24 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,1);
25 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,2);
26 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (2,1);
27
28 DELETE FROM CURSO WHERE CODIGOCURSO=1;
```

Após a execução, com sucesso, do trecho entre as linhas 1 e 26, se executarmos o comando expresso na linha 28, quantas linhas serão removidas do SGBD?

A alternativa **"D "** está correta.

De fato, a chave estrangeira declarada na linha 15 foi criada de maneira a permitir a deleção em cascata. Ao executar o comando da linha 28, o SGBD eliminará tanto os dois registros da tabela CURSODISCIPLINA (cujo valor de CODIGOCURSO é igual a 1) quanto o registro referente ao curso Sistemas de Informação.

MÓDULO 4

- Empregar comandos de controle de transação

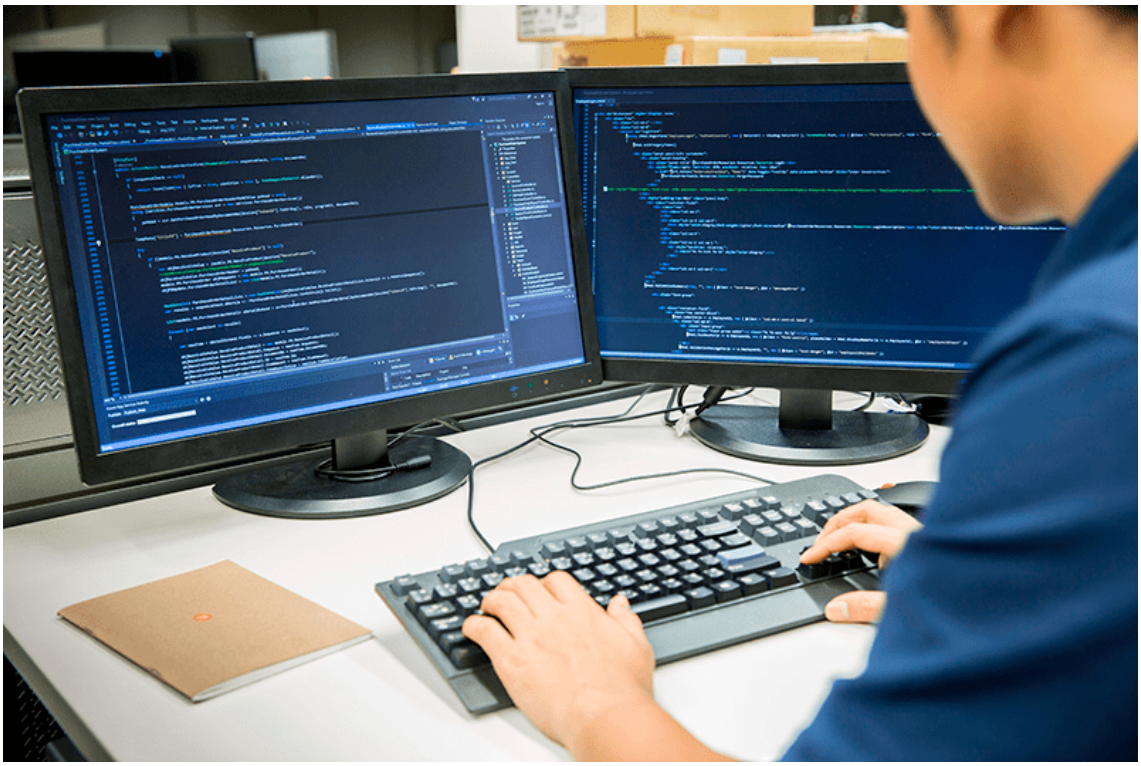
TRANSAÇÕES EM BANCO DE DADOS

Ao longo do nosso estudo, aprendemos uma série de comandos SQL envolvendo desde a criação de tabelas até operações de manipulação de dados, tais como inserções, atualizações e exclusões. Basicamente, um comando era codificado e submetido ao SGBD PostgreSQL, que em seguida o executava e devolvia algum resultado.

Perceba que, sob o contexto dos exemplos que estudamos, poderíamos concluir que, de certa maneira, havia somente um usuário acessando a totalidade dos recursos do SGBD.

No entanto, em um ambiente de produção, o SGBD gerencia centenas de requisições das aplicações. Com isso, concluímos que há acesso simultâneo a vários recursos que são gerenciados pelo SGBD.

Ao mesmo tempo, o usuário de uma aplicação que faz uso de recursos de um banco de dados, normalmente, está preocupado com o resultado dos processos de negócio que estão automatizados.



Fonte: PIYAWAT WONGOPASS/Shutterstock

Para exemplificar, digamos que um sistema acadêmico disponibiliza a inscrição em diversas disciplinas; determinado aluno, após consultar a oferta e escolher um conjunto de matérias para inscrição, tem expectativa de conseguir inscrever-se em todas as disciplinas alvo da escolha.

No entanto, sob o ponto de vista do SGBD, prover a inscrição em todas as disciplinas escolhidas pelo estudante requer a execução, na totalidade, de diversas instruções de inserção de dados em alguma tabela. Além disso, deve existir cuidado especial para, por exemplo, inibir inscrição em disciplina caso não haja mais disponibilidade de vagas.

📢 ATENÇÃO

É uma situação típica sobre a qual o SGBD precisa prover uma forma de realizar diversas operações como uma unidade lógica de processamento. Vamos aprender, então, que essa unidade de processamento é denominada transação.

Em sistemas de banco de dados, uma transação corresponde a um programa em execução que forma uma unidade de trabalho.

Os limites de uma transação são especificados por meio dos comandos *begin transaction* (que indica o início de uma transação) e *end transaction* (que indica o término de uma transação)

em um programa de aplicação.

Ainda, se a transação não atualiza o banco de dados, é denominada somente de leitura; caso contrário, é chamada de leitura-gravação.

Para fins didáticos, um modelo simplificado do banco de dados - coleção de itens nomeados - é usado para estudo dos conceitos de processamento de transações. Cada item de dados pode representar um registro, bloco ou valor de coluna.

Em se tratando de SGBDs multiusuários, várias transações podem ser executadas simultaneamente. No entanto, caso essa execução ocorra de maneira descontrolada, poderão surgir problemas de inconsistências, tais como:

ATUALIZAÇÃO PERDIDA

ATUALIZAÇÃO TEMPORÁRIA

RESUMO INCORRETO

LEITURA NÃO REPETITIVA

ATUALIZAÇÃO PERDIDA

Quando duas transações que acessam os mesmos itens de dados têm operações intercaladas de modo a tornar incorretos alguns itens do banco de dados.

ATUALIZAÇÃO TEMPORÁRIA

Quando uma transação atualiza um item do banco de dados e, depois, falha por algum motivo, enquanto, nesse meio tempo, o item é lido por outra transação antes de ser alterado de volta para seu valor original.

RESUMO INCORRETO

Quando uma transação calcula uma função de resumo de agregação em uma série de itens enquanto outras transações atualizam alguns desses itens.

LEITURA NÃO REPETITIVA

Quando uma transação lê o mesmo item duas vezes e o item é alterado por outra transação entre as duas leituras.

É importante sabermos que, quando o SGBD processa uma transação, todas as operações que a formam devem ser completadas com sucesso para, somente depois, haver a gravação permanente das alterações no banco de dados. Além disso, caso haja falha em uma ou mais operações de uma transação, as demais operações não devem ser executadas, e a transação deve ser cancelada.

Se uma transação for cancelada, deve ser executado um processo denominado ***rollback*** (Rolar para trás) , o qual força o SGBD a trazer de volta os valores antigos dos registros antes da transação ter iniciado. Finalmente, caso a transação seja executada com sucesso, as atualizações devem ser confirmadas por meio do comando ***commit*** (Confirmação) .

Estas são as falhas que podem ocorrer durante o processamento de uma transação:

Falha do computador
Erro de transação ou sistema
Condições de exceção detectadas pela transação
Falha de disco, problemas físicos e catástrofes

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

A seguir, conheceremos as propriedades das transações.

PROPRIEDADES DE UMA TRANSAÇÃO

Com objetivo de garantir a integridade dos dados contidos no banco de dados, o SGBD mantém um conjunto de propriedades das transações, denominado **ACID**, que representam as características de atomicidade, consistência, isolamento e durabilidade, respectivamente.

ACID

Do Inglês, *Atomicity, Consistency, Isolation, Durability*.

Atomicidade	A transação precisa ser realizada completamente ou não realizada
Consistência	A transação deve levar o banco de dados de um estado consistente para outro
Isolamento	A execução de uma transação não deve ser interferida por quaisquer outras transações
Durabilidade	As mudanças no banco de dados em função da confirmação da transação devem persistir

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

ESTADOS DE UMA TRANSAÇÃO

Uma transação pode passar pelos seguintes estados ao longo do seu processamento:

ATIVO

Ocorre imediatamente após o início da transação, podendo executar operações de leitura e gravação.

PARCIALMENTE CONFIRMADO

Quando a transação termina.

CONFIRMADO

Após verificação bem-sucedida, as mudanças são gravadas permanentemente no banco de dados.

FALHA

Se uma das verificações falhar ou se a transação for abortada durante seu estado ativo.

CONFIRMADO

Transação sai do sistema.

Para poder recuperar-se de falhas que afetam transações, normalmente, o SGBD mantém um *log* para registrar todas as operações de transação que afetam os vários itens do banco de dados. As entradas em um registro de *log* de uma determinada transação possuem informações de valores antigos e novos do item de dados do banco de dados, bem como se a transação foi concluída com sucesso ou se foi abortada.

TRANSAÇÕES NO POSTGRESQL

De maneira geral, uma transação no PostgreSQL possui a estrutura a seguir:

BEGIN-- início da transação
-- comandos
COMMIT-- transação confirmada
ROLLBACK-- transação cancelada
END-- mesma função do COMMIT

No entanto, nos SGBDs, a inicialização de uma transação ocorre implicitamente quando executamos alguns comandos.

Vamos estudar um exemplo?

Seja a tabela CURSO, conforme a seguir:

CURSO		
CODIGOCURSO	int	PK
NOME	varchar(90)	
DATACRIACAO	date	N

Fonte: Autor

📷 Tabela CURSO.

A execução de um INSERT na tabela ocorre dentro do contexto implícito de uma transação:

-- BEGIN implícito

INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (5,'Engenharia de Computação',NULL);

-- COMMIT implícito

Estudamos que, quando uma transação é desfeita, qualquer operação que faz parte da transação deve ser cancelada. Vamos, então, ver como podemos fazer isso no PostgreSQL. Veja o exemplo a seguir, construído com objetivo de inserir um registro na tabela CURSO e, em seguida, indicar que a operação de inserção ser desfeita pelo SGBD:

```
1 SELECT * FROM CURSO; -- Dados atuais
2 BEGIN;
3 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (6,'Psicologia',NULL);
4 SELECT * FROM CURSO; -- Dados após inserção
5 ROLLBACK;
6 END;
7 SELECT * FROM CURSO; -- Dados após desfazer a transação
```

Fonte: O autor

Após o processamento do comando da linha 1, visualizaremos o conteúdo da tabela CURSO da seguinte maneira:

	codigocurso	nome	datacriacao
1	2	Medicina	1990-05-10
2	1	Sistemas de Informação	1999-06-19
3	3	Nutrição	2012-02-19
4	4	Pedagogia	1999-06-19

Fonte: O autor

📷 Tabela CURSO após a execução do comando da linha 1.

Após o processamento dos comandos da linha 2 à linha 4, que já representam a inserção de um registro na tabela CURSO sob o contexto de uma transação explícita, teremos este

resultado:

	codigocurso	nome	datacriacao
1	2	Medicina	1990-05-10
2	1	Sistemas de Informação	1999-06-19
3	3	Nutrição	2012-02-19
4	4	Pedagogia	1999-06-19
5	6	Psicologia	[NULL]

Fonte: O autor

📷 Tabela CURSO após a execução do comando da linha 4.

Finalmente, após o processamento dos comandos da linha 5 à linha 7, onde a transação é desfeita, teremos o resultado conforme a tabela a seguir:

	codigocurso	nome	datacriacao
1	2	Medicina	1990-05-10
2	1	Sistemas de Informação	1999-06-19
3	3	Nutrição	2012-02-19
4	4	Pedagogia	1999-06-19

Fonte: O autor

📷 Tabela CURSO após a execução do comando da linha 7.

Perceba que o comando da linha 2 (BEGIN) iniciou explicitamente uma transação, a qual foi abortada quando da execução do comando da linha 5 (ROLLBACK).

📢 ATENÇÃO

No exemplo anterior, programamos uma transação que envolveu somente uma operação, a saber, inserção de uma linha na tabela CURSO. No entanto, uma transação pode envolver diversas linhas de uma tabela do banco de dados.

Vamos, então, programar uma transação que consistirá em duas operações de atualização envolvendo a tabela que contém as disciplinas, apresentada conforme a seguir:

DISCIPLINA		
CODIGODISCIPLINA	int	PK
NOME	varchar(90)	
CARGAHORARIA	int	

Fonte: O autor

Tabela DISCIPLINA.

Inicialmente, vamos listar o conteúdo da tabela DISCIPLINA. Podemos, então, usar o comando a seguir:

SELECT * FROM DISCIPLINA;

Após o processamento do comando, teremos o seguinte resultado:

	codigodisciplina	nome	cargahoraria
1	1	Leitura e Produção de Textos	60
2	2	Redes de Computadores	60
3	3	Computação Gráfica	40
4	4	Anatomia	60

Fonte: O autor

Conteúdo da tabela DISCIPLINA com os dados originais.

Agora, nossa intenção é alterar a carga horária das disciplinas de acordo com os critérios a seguir:

Disciplinas que possuem 60 horas: aumento em 20%
Disciplinas que possuam 40 horas: aumento em 10%

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

O código a seguir expressa uma transação envolvendo operações de atualização de dados na tabela DISCIPLINA:

```
1 BEGIN;
2 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.2 WHERE CARGAHORARIA=60;
3 SELECT * FROM DISCIPLINA;
4 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.1 WHERE CARGAHORARIA=40;
5 SELECT * FROM DISCIPLINA;
6 COMMIT;
```

Fonte: O autor

Após a execução do trecho entre as linhas 1 e 3, o conteúdo da tabela disciplina estará conforme a seguir:

	codigodisciplina	nome	cargahoraria
1	3	Computação Gráfica	40
2	1	Leitura e Produção de Textos	72
3	2	Redes de Computadores	72
4	4	Anatomia	72

Fonte: O autor

Conteúdo da tabela DISCIPLINA após a execução da linha 3.

Após a execução da transação, o conteúdo da tabela disciplina estará conforme a seguir:

	codigodisciplina	nome	cargahoraria
1	1	Leitura e Produção de Textos	72
2	2	Redes de Computadores	72
3	4	Anatomia	72
4	3	Computação Gráfica	44

Fonte: O autor

Conteúdo da tabela DISCIPLINA após o término da transação.

Outro ponto interessante no projeto de transações é a utilização de pontos de salvamento (SAVEPOINT). Observe o exemplo a seguir:

```
1 BEGIN;
2 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.2 WHERE CARGAHORARIA=60;
3 SELECT * FROM DISCIPLINA;
4 SAVEPOINT CARGA60;
5 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.1 WHERE CARGAHORARIA=40;
6 ROLLBACK TO CARGA60;
7 SELECT * FROM DISCIPLINA;
8 COMMIT;
```

Fonte: O autor

Na linha 4, adicionamos um SAVEPOINT denominado CARGA60. Quando a linha 6 for executada, o SGBD vai desfazer a operação de UPDATE da linha 5.

UM POUCO MAIS SOBRE ATUALIZAÇÃO TEMPORÁRIA

Estudamos que uma transação não deve atrapalhar o andamento de outra. Pense na execução da nossa última transação, que envolveu dois comandos de atualização na tabela DISCIPLINA.

Vimos que, logo após a execução da linha 3, a única disciplina que não sofreu alteração foi a de Computação Gráfica. Perceba que o SELECT da linha 3 está sendo executado no contexto da transação.

No entanto, o que aconteceria se tivéssemos em paralelo outra aplicação que submetesse consulta para acessar os registros da tabela DISCIPLINA no mesmo momento da execução da linha 3 da transação?

RESPOSTA

A consulta em questão enxergaria os dados “originais”, sem quaisquer alterações. Por qual razão? Para não haver o problema da atualização temporária. Queremos dizer que, se a transação fosse desfeita por qualquer motivo, o UPDATE da linha 2 seria também desfeito.

UM POUCO MAIS SOBRE TRANSAÇÃO DE LEITURA

Vimos que uma transação que não modifica dados é denominada transação somente de leitura (READ ONLY). Caso contrário, é denominada leitura-gravação (READ WRITE).

Para especificar o tipo de transação, usaremos o comando SET TRANSACTION <TIPOTRANSAÇÃO>. No PostgreSQL, quando iniciamos uma transação, o padrão é READ WRITE.

Vamos analisar um exemplo envolvendo uma transação READ ONLY:

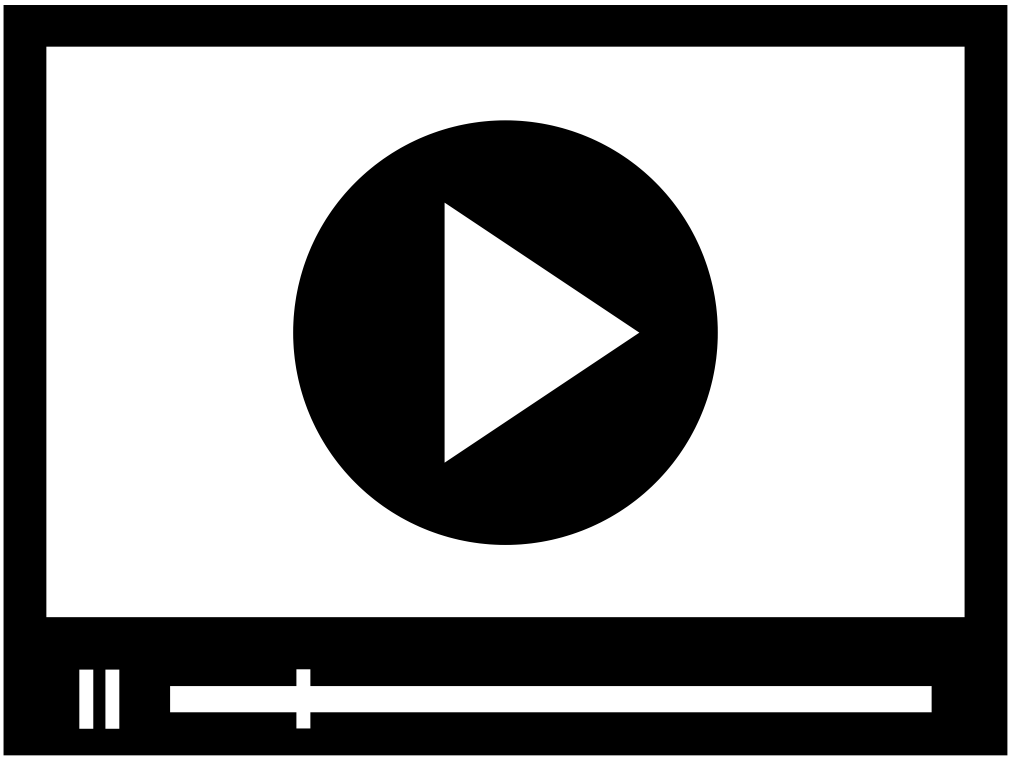
```
1 BEGIN;
2 SET TRANSACTION READ ONLY;
3 UPDATE DISCIPLINA SET CARGAHORARIA=80;
4 ROLLBACK;
```

Fonte: Autor

Na transação acima, propositalmente, inserimos um comando de atualização em uma transação que não permite essa categoria de comando. Logo, após a execução da linha 3, o SGBD retornará uma mensagem informando ao usuário que não é possível executar comando de atualização em uma transação do tipo somente leitura.

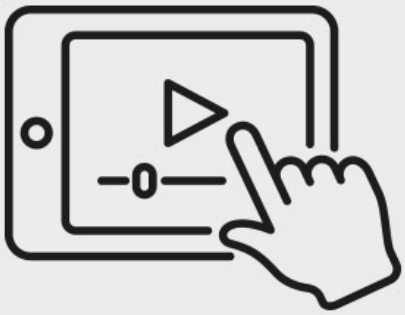
Ao longo deste módulo, aprendemos o conceito de transação, que representa uma sequência de comandos que devem ser executados na totalidade, ou, caso contrário, desfeitos.

Vimos que, por padrão, o SGBD PostgreSQL implicitamente gera uma transação quando submetemos algum comando de inserção, atualização ou remoção de dados. Por fim, aprendemos comandos para gerenciar transações no PostgreSQL.



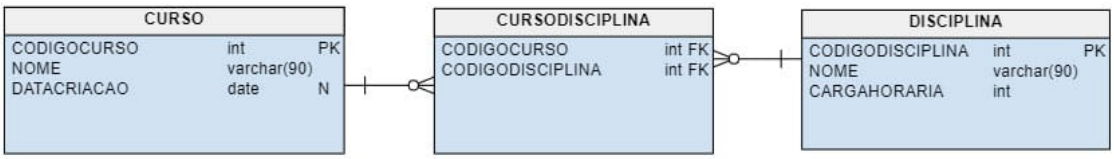
COMANDOS DE CONTROLE DE TRANSAÇÕES

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. A RESPEITO DE TRANSAÇÕES NO POSTGRESQL E, CONSIDERANDO O MODELO A SEGUIR, ASSINALE A PROPOSIÇÃO VERDADEIRA:



A) Ao executar o comando DELETE FROM CURSODISCIPLINA(; o PostgreSQL não executa uma transação.

B) O comando DELETE FROM CURSODISCIPLINA(; pode ser executado sem erro em uma transação PostgreSQL do tipo READ ONLY.

C) O comando DELETE FROM CURSODISCIPLINA(; pode ser executado sem erro em uma transação PostgreSQL do tipo READ WRITE.

D) O comando SELECT * FROM CURSODISCIPLINA(; não pode ser executado em uma transação PostgreSQL do tipo READ ONLY.

2. SUPONHA QUE UM PROFISSIONAL PROGRAMOU NO POSTGRESQL OS SEGUINTES COMANDOS EM UMA TABELA DENOMINADA

EMPREGADO:

```
1 BEGIN;
2 UPDATE funcionario SET salario = salario + 2000.00 WHERE nome = 'Maria';
3
4 UPDATE funcionario SET salario = salario + 1000.00 WHERE nome = 'Paulo';
5 COMMIT;
```

SUPONHA TAMBÉM QUE, APÓS A EXECUÇÃO DA LINHA 2, O PROFISSIONAL PERCEBEU QUE NÃO DEVERIA TER AUMENTADO O SALÁRIO DE MARIA NESSE VALOR. QUAL COMANDO É ADEQUADO ADICIONAR À LINHA 3 PARA DESFAZER ESSA OPERAÇÃO?

A) DELETE;

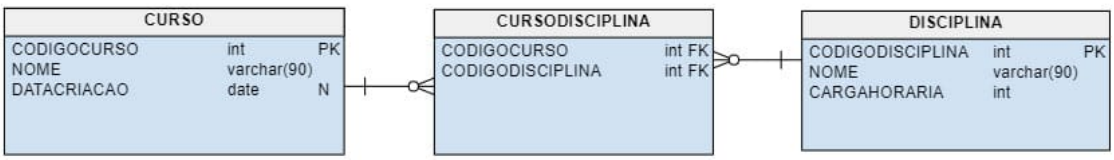
B) COMMIT;

C) UPDATE;

D) ROLLBACK;

GABARITO

1. A respeito de transações no PostgreSQL e, considerando o modelo a seguir, assinale a proposição verdadeira:



A alternativa **"C "** está correta.

De fato, se uma transação no PostgreSQL é definida como READ WRITE, ela aceita comandos de inserção, atualização e remoção de dados. Logo, a instrução que contém o comando DELETE poderá ser executada sem erro.

2. Suponha que um profissional programou no PostgreSQL os seguintes comandos em uma tabela denominada empregado:

```
1 BEGIN;
2 UPDATE funcionario SET salario = salario + 2000.00 WHERE nome = 'Maria';
3
4 UPDATE funcionario SET salario = salario + 1000.00 WHERE nome = 'Paulo';
5 COMMIT;
```

Suponha também que, após a execução da linha 2, o profissional percebeu que não deveria ter aumentado o salário de Maria nesse valor. Qual comando é adequado adicionar à linha 3 para desfazer essa operação?

A alternativa **"D "** está correta.

De fato, como a transação não foi concluída, é possível desfazer a operação da linha 2, bastando para isso adicionar o comando ROLLBACK. Após isso, a tabela funcionário ficará com os registros iguais à situação imediatamente anterior à execução da transação.

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Ao longo do nosso estudo, fizemos uma introdução aos recursos do SGBD PostgreSQL, envolvendo características desse SGBD, bem como sua instalação.

Foram apresentados comandos, classificados como DDL, para a criação e a alteração de tabelas. Em seguida, conhecemos diversos comandos SQL para manipulação de linhas em tabelas. Tais comandos, classificados como DML, são úteis para inserção, alteração e remoção de dados.

Finalizamos com uma breve contextualização a respeito do conceito, uso e importância das transações, com destaque aos comandos do PostgreSQL utilizados para esse fim.

Para ouvir um *podcast* sobre o assunto, acesse a versão online deste conteúdo.

AVALIAÇÃO DO TEMA

REFERÊNCIAS

DBEAVER COMMUNITY. Consultado em meio eletrônico em: 30 mai. 2020.

ELMASRI, R.; NAVATHE, S. **Sistemas de Banco de Dados**. 7. ed. São Paulo: Pearson, 2019.

MANZANO, J.A.M.G., **Microsoft SQL Server 2016 Express Edition Interativo**. 1. ed. São Paulo: Saraiva, 2017.

POSTGRESQL. **PostgreSQL 12.3 Documentation**. Consultado em meio eletrônico em: 30 mai. 2020.

POSTGRESQL. **Create Table. Consultado em meio eletrônico em: 30 mai. 2020.**

POSTGRESQL. Data Type. Consultado em meio eletrônico em: 30 mai. 2020.

POSTGRESQL. Delete. Consultado em meio eletrônico em: 30 mai. 2020.

POSTGRESQL. Download. Consultado em meio eletrônico em: 30 mai. 2020.

POSTGRESQL. Insert. Consultado em meio eletrônico em: 30 mai. 2020.

POSTGRESQL. Ubuntu. Consultado em meio eletrônico em: 30 mai. 2020.

POSTGRESQL. Update. Consultado em meio eletrônico em: 30 mai. 2020.

EXPLORE+

Para aprofundar seus conhecimentos sobre o assunto deste tema, leia:

> **CHAMBERLIN, D. D. Early History of SQL *In*: IEEE Annals of the History of Computing n 4, 2012. Como vimos, a linguagem SQL tornou-se um padrão para uso em sistemas gerenciadores de bancos de dados relacionais. O artigo indicado é um interessante material sobre a história da SQL.**

CONTEUDISTA

Nathielly de Souza Campos

🔗 CURRÍCULO LATTES